UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

DESARROLLO DE UN MÓDULO DE ADAPTACIÓN EN TIEMPO DE EJECUCIÓN PARA UN REPRODUCTOR DE PROCESOS DE APRENDIZAJE

TUTOR: TELMO ZARRAONANDIA AYO

AUTOR: PEDRO SANTOS MORENO FEBRERO 2010

AGRADECIMIENTOS

A mis padres y hermanas, por aguantarme y apoyarme durante todos estos años, y que han estado ahí tanto en los momentos malos como en los buenos.

A mis compañeros, algunos de ellos amigos, gracias a los cuales la realización de prácticas (esas tardes hasta las 9 en las Linux), el estudio (esas semanas encerrados en la biblioteca con interminables descansos) y, en definitiva, mi estancia en la Universidad (incluyendo las charlas y el mus de después de la comida, la práctica de fútbol y la asistencia al gimnasio), ha sido mucho más agradable y llevadera.

A mis amigos de Calalberche, por permitirme desconectar del mundo universitario los fines de semana (a veces empezando los jueves).

A mi tutor, por las horas y horas empleadas para entender el funcionamiento de CopperCore y por su dedicación y entrega en la mejora de esta memoria, y por la confianza depositada en mí.

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS 1
ÍNDICE DE FIGURAS 3
ÍNDICE DE TABLAS 5
1 INTRODUCCIÓN 9
1.1 CONTEXTO 9
1.2 PLANTEAMIENTO DEL PROBLEMA 10
1.3 OBJETIVOS DEL PROYECTO 11
1.4 ESTRUCTURA DEL DOCUMENTO 12
2 METODOLOGÍA DE TRABAJO Y PLANIFICACIÓN 13
2.1 METODOLOGÍA DE TRABAJO 13
2.2 FASES DEL PROYECTO 15
2.3 PLANIFICACIÓN 18
2.4 HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS 21
2.4.1 J2EE 21
2.4.2 ANT 22
2.4.3 ECLIPSE 23
2.4.4 JDK 1.4 23
2.4.5 XML 24
2.4.6 MICROSOFT OFFICE 2007 24
2.5 COSTES GENERADOS 25
3 ESTADO DEL ARTE 29
3.1 ADAPTACIONES EN PROCESOS EDUCATIVOS SUPERVISADOS POR INSTRUCTOR 30
3.1.1 CARACTERÍSTICAS DE LAS ADAPTACIONES 31
3.2 SOFTWARE DE PROCESOS EDUCATIVOS: ENFOQUES DE DISEÑO 33
3.2.1 SISTEMAS DE GESTIÓN DEL APRENDIZAJE (LMS) 33
3.2.1.1 ESTÁNDARES 34
3.3 LENGUAJES DE MODELADO EDUCATIVO 37
3.3.1 IMS LEARNING DESIGN 38
3.3.1.1 REQUISITOS DE DISEÑO 39
3.3.1.2 ESPECIFICACIÓN DE IMS LD 39
3.3.1.3 NIVELES DE ESPECIFICACIÓN DE IMS LD 41
3.3.1.4 AUTORÍA DE CONTENIDOS 43
3.3.1.5 REPRODUCTORES 45
3.4 MODELO DE ADAPTACIÓN PARA IMS LD: ADAPTATION POKES 48
3.4.1 TIPOS DE ADAPTACIONES 49

3.4.2 TIPOS DE ADAPTATION POKES 49
4 ANÁLISIS 51
4.1 CARACTERÍSTICAS DEL ENTORNO 51
4.2 ESPECIFICACIÓN DE REQUISITOS 52
4.2.1 REQUISITOS FUNCIONALES 53
4.2.1.1 REQUISITOS FUNCIONALES DE MODIFICACIÓN 53
4.2.1.2 REQUISITOS FUNCIONALES DE ADICIÓN 56
4.2.1.3 REQUISITOS FUNCIONALES DE BORRADO 58
4.2.1.4 OTROS REQUISITOS FUNCIONALES 59
4.2.2 REQUISITOS NO FUNCIONALES 60
4.2.2.1 REQUISITOS NO FUNCIONALES OPERATIVOS 60
4.2.2.2 REQUISITOS NO FUNCIONALES DE SEGURIDAD 61
4.3 ESPECIFICACIÓN DE CASOS DE USO 61
5 DISEÑO 63
5.1 ARQUITECTURA 63
5.2 DISEÑO DETALLADO 65
5.2.1 RELACIÓN ENTRE LOS COMPONENTES DE LA ARQUITECTURA 65
5.2.2 COMPONENTE COPPERCORE 66
5.2.3 MÓDULO DE ADAPTACIÓN 72
6 IMPLEMENTACIÓN 75
6.1 PAQUETE CLIENTS 75
6.2 PAQUETE VALIDATOR 75
6.3 PAQUETE COMPONENT 76
7 MANUAL DE USUARIO 81
7.1 CONFIGURACIÓN DEL ENTORNO 81
7.2 EJECUCIÓN DE COPPERCORE 82
7.3 CONSOLA DE GESTIÓN DE COPPERCORE 87
8 EVALUACIÓN 91
8.1 PLAN DE PRUEBAS 91
8.1.1 CASOS DE PRUEBA DE MODIFICACIÓN 92
8.1.2 CASOS DE PRUEBA DE ADICIÓN 99
8.1.3 CASOS DE PRUEBA DE BORRADO 103
8.2 MATRIZ DE TRAZABILIDAD 106
9 CONCLUSIONES 109
ACRÓNIMOS 111
BIBLIOGRAFÍA 115
GLOSARIO DE TÉRMINOS 117

ÍNDICE DE FIGURAS

Ilustración 2.1 Modelo de ciclo de vida en cascada realimentado 14
Ilustración 2.2 Fases del proyecto 17
Ilustración 2.3 Diagrama de Gantt con la planificación inicial 19
Ilustración 2.4 Diagrama de Gantt con la planificación real 20
Ilustración 2.5 Servicios ofrecidos por J2EE 21
Ilustración 2.6 Modelo de Aplicación Empresarial de J2EE 22
Ilustración 3.1 Estructura de una UOL 38
Ilustración 3.2 Modelo Conceptual de Learning Design 40
Ilustración 3.3 Editor de IMS LD del proyecto RELOAD 44
Ilustración 3.4 Editor de IMS LD CopperAuthor 44
Ilustración 3.5 Reproductor de Learning Design de CopperCore 46
Ilustración 3.6 Reproductor de Learning Design del proyecto RELOAD 47
Ilustración 3.7 Reproductor de Learning Design de SLED 48
Ilustración 3.8 Diagrama con los componentes de un Adaptation Poke 50
Ilustración 4.1 Diagrama de casos de uso 62
Ilustración 5.1 Arquitectura del sistema 64
Ilustración 5.2 Diagrama de Secuencia para la publicación de un Adaptation Poke 65
Ilustración 5.3 Estructura de CopperCore 67
Ilustración 5.4 Interfaces de CopperCore 68
Ilustración 5.5 Modelo UML con la arquitectura de implementación de servicios 69
Ilustración 5.6 Diagrama UML de componentes de las interfaces de CopperCore 71
Ilustración 5.7 Diagrama de clases del componente Clients 72
Ilustración 5.8 Diagrama de clases de los componentes Component y Validator 73
Ilustración 5.9 Diagrama de clases del componente ejb 74
Ilustración 7.1 Pantalla de Propiedades del sistema 81
Ilustración 7.2 Pantalla de Variables de entorno 82
Ilustración 7.3 Pantalla ejecutar 83
Ilustración 7.4 Pantalla JBoss Application Server Running CopperCore 83
Ilustración 7.5 Pantalla de la consola de gestión de CopperCore 84
Ilustración 7.6 Pantalla Publisher 84
Ilustración 7.7 Pantalla de validación 85
Ilustración 7.8 Pantalla del reproductor 85

Ilustración 7.9 Pantalla con la UOL desplegada 86
Ilustración 7.10 Pantalla de Clicc con los comandos disponibles 87
Ilustración 7.11 Pantalla de Clicc con los comandos createuser y dir 87
Ilustración 7.12 Pantalla de Clicc con los comandos cd y createrun 88
Ilustración 7.13 Pantalla de Clicc con el comando addusertorun 89
Ilustración 7.14 Pantalla de Clicc con los comandos listroles, addusertorole y setactiverole 89
Ilustración 7.15 Pantalla de Clicc con el comando uploadpoke 90
Ilustración 7.16 Pantalla con cambios en la UOL 90

ÍNDICE DE TABLAS

Tabla 2.1 Comparación entre días estimados y días reales 18
Tabla 2.2 Precio por hora de cada trabajador 25
Tabla 2.3 Días trabajados de cada mes en cada fase 25
Tabla 2.4 Días empleados por rol y mes 26
Tabla 2.5 Horas totales trabajadas por rol 26
Tabla 2.6 Coste total del personal implicado en el proyecto 26
Tabla 2.7 Coste del equipamiento hardware 27
Tabla 2.8 Coste del equipamiento software 27
Tabla 2.9 Coste de los recursos fungibles 27
Tabla 2.10 Gasto total sin IVA 27
Tabla 2.11 Coste total del Proyecto Fin de Carrera 28
Tabla 4.1 Requisito Funcional RF-001 53
Tabla 4.2 Requisito Funcional RF-002 53
Tabla 4.3 Requisito Funcional RF-003 53
Tabla 4.4 Requisito Funcional RF-004 54
Tabla 4.5 Requisito Funcional RF-005 54
Tabla 4.6 Requisito Funcional RF-006 54
Tabla 4.7 Requisito Funcional RF-007 54
Tabla 4.8 Requisito Funcional RF-008 54
Tabla 4.9 Requisito Funcional RF-009 54
Tabla 4.10 Requisito Funcional RF-010 55
Tabla 4.11 Requisito Funcional RF-011 55
Tabla 4.12 Requisito Funcional RF-012 55
Tabla 4.13 Requisito Funcional RF-013 55
Tabla 4.14 Requisito Funcional RF-014 56
Tabla 4.15 Requisito Funcional RF-015 56
Tabla 4.16 Requisito Funcional RF-016 56
Tabla 4.17 Requisito Funcional RF-017 56
Tabla 4.18 Requisito Funcional RF-018 56
Tabla 4.19 Requisito Funcional RF-019 57
Tabla 4.20 Requisito Funcional RF-020 57
Tabla 4.21 Requisito Funcional RF-021 57

Tabla 4.22 Requisito Funcional RF-022 58
Tabla 4.23 Requisito Funcional RF-023 58
Tabla 4.24 Requisito Funcional RF-024 58
Tabla 4.25 Requisito Funcional RF-025 58
Tabla 4.26 Requisito Funcional RF-026 58
Tabla 4.27 Requisito Funcional RF-027 59
Tabla 4.28 Requisito Funcional RF-028 59
Tabla 4.29 Requisito Funcional RF-029 59
Tabla 4.30 Requisito Funcional RF-030 59
Tabla 4.31 Requisito No Funcional Operativo RNFO-001 60
Tabla 4.32 Requisito No Funcional Operativo RNFO-002 60
Tabla 4.33 Requisito No Funcional Operativo RNFO-003 60
Tabla 4.34 Requisito No Funcional Operativo RNFO-004 60
Tabla 4.35 Requisito No Funcional Operativo RNFO-005 61
Tabla 4.36 Requisito No Funcional de Seguridad RNFS-001 61
Tabla 4.37 Caso de uso CU-001 62
Tabla 8.1 Caso de prueba CP-001 92
Tabla 8.2 Caso de prueba CP-002 92
Tabla 8.3 Caso de prueba CP-003 93
Tabla 8.4 Caso de prueba CP-004 93
Tabla 8.5 Caso de prueba CP-005 93
Tabla 8.6 Caso de prueba CP-006 94
Tabla 8.7 Caso de prueba CP-007 94
Tabla 8.8 Caso de prueba CP-008 94
Tabla 8.9 Caso de prueba CP-009 95
Tabla 8.10 Caso de prueba CP-010 95
Tabla 8.11 Caso de prueba CP-011 95
Tabla 8.12 Caso de prueba CP-012 96
Tabla 8.13 Caso de prueba CP-013 96
Tabla 8.14 Caso de prueba CP-014 96
Tabla 8.15 Caso de prueba CP-015 97
Tabla 8.16 Caso de prueba CP-016 97
Tabla 8.17 Caso de prueba CP-017 97
Tabla 8.18 Caso de prueba CP-018 98

Tabla 8.19 Caso de prueba CP-019 98
Tabla 8.20 Caso de prueba CP-020 98
Tabla 8.21 Caso de prueba CP-021 99
Tabla 8.22 Caso de prueba CP-022 99
Tabla 8.23 Caso de prueba CP-023 99
Tabla 8.24 Caso de prueba CP-024 100
Tabla 8.25 Caso de prueba CP-025 100
Tabla 8.26 Caso de prueba CP-026 100
Tabla 8.27 Caso de prueba CP-027 101
Tabla 8.28 Caso de prueba CP-028 101
Tabla 8.29 Caso de prueba CP-029 101
Tabla 8.30 Caso de prueba CP-030 102
Tabla 8.31 Caso de prueba CP-031 102
Tabla 8.32 Caso de prueba CP-032 102
Tabla 8.33 Caso de prueba CP-033 103
Tabla 8.34 Caso de prueba CP-034 103
Tabla 8.35 Caso de prueba CP-035 103
Tabla 8.36 Caso de prueba CP-036 104
Tabla 8.37 Caso de prueba CP-037 104
Tabla 8.38 Caso de prueba CP-038 104
Tabla 8.39 Caso de prueba CP-039 105
Tabla 8.40 Caso de prueba CP-040 105
Tabla 8.41 Caso de prueba CP-041 105
Tabla 8.42 Caso de prueba CP-042 106
Tabla 8.43 Caso de prueba CP-043 106
Tabla 8.44 Matriz de trazabilidad para los casos de prueba CP-001 a CP-007..... 106
Tabla 8.45 Matriz de trazabilidad para los casos de prueba CP-008 a CP-014..... 107
Tabla 8.46 Matriz de trazabilidad para los casos de prueba CP-015 a CP-021..... 107
Tabla 8.47 Matriz de trazabilidad para los casos de prueba CP-022 a CP-028..... 107
Tabla 8.48 Matriz de trazabilidad para los casos de prueba CP-029 a CP-035..... 108
Tabla 8.49 Matriz de trazabilidad para los casos de prueba CP-036 a CP-042..... 108
Tabla 8.50 Matriz de trazabilidad para el caso de prueba CP-043 108

1 INTRODUCCIÓN

La finalidad principal de este Proyecto Fin de Carrera de la titulación de Ingeniería Informática es la de desarrollar un mecanismo que permita extender un motor de ejecución de procesos de aprendizaje, concretamente CopperCore, con el fin de implementar adaptaciones de procesos de aprendizaje especificados mediante IMS LD en tiempo de ejecución. Este capítulo ofrece una visión general de todo el documento.

1.1 CONTEXTO

Este Proyecto Fin de Carrera se enmarca dentro del ámbito de los procesos de aprendizaje asistidos por ordenador en los que es necesaria la presencia de un tutor que los supervise. La figura de este tutor es de suma importancia debido a que ejerce un rol de guía sobre los alumnos para garantizar que se cumpla con la planificación establecida y para ofrecer asistencia pedagógica en caso de que sea necesario. Este tipo de proceso de aprendizaje es un modelo que se ha estado siguiendo desde hace décadas en la enseñanza a distancia, si bien con el auge de las nuevas tecnologías ha experimentado un auge espectacular.

A la hora de analizar las soluciones informáticas que se han desarrollado durante las últimas décadas para dar soporte a este tipo de procesos de aprendizaje, nos vamos a centrar en el seguido por los *Learning Management Systems* (LMS) o Sistemas de Gestión del Aprendizaje. Con este enfoque lo que se persigue es, desde un punto de vista comercial, desarrollar entornos virtuales que proporcionen los servicios necesarios para el desarrollo de experiencias educativas.

Los LMS son aplicaciones *software* que automatizan las tareas de administración, gestión y monitorización de procesos de aprendizaje. De manera paralela al desarrollo de este tipo de aplicaciones surgió la necesidad de establecer estándares para el desarrollo de los contenidos educativos, de forma que se garantizase su interoperabilidad y se asegurase la rentabilidad de las inversiones realizadas por los fabricantes.

El estándar en el que se basa este Proyecto Fin de Carrera es el desarrollado por el IMS *Global Consistorium Inc.*, más concretamente IMS *Learning Design*. IMS *Learning Design* es un lenguaje de modelado educativo (EML, siglas en inglés de *Educational Modelling Language*), el más completo y aceptado en la actualidad, en el que, además de detallar los componentes y recursos que van a ser empleados en un proceso educativo, se describen explícitamente las actividades que se llevarán a cabo, así como el uso que hará cada una de ellas de los recursos disponibles, los roles de los participantes de las actividades, etc. Además, IMS *Learning Design* permite la creación de descripciones completas, abstractas y portables desde el punto de vista pedagógico de forma que puedan ser puestas en práctica a través de motores de ejecución compatibles con la especificación.

Este estándar se basa en el uso de Unidades de Aprendizaje (UOL, siglas en inglés de *Unit Of Learning*), que son ficheros que contienen un manifiesto, en el que se incluye toda la información para llevar a cabo un proceso educativo, y todos los recursos necesarios. El formato de especificación del manifiesto es el lenguaje XML.

Una vez editada una UOL, existen diversos programas que se encargan de su interpretación y de su representación en una interfaz con el fin de poder desarrollar las actividades propuestas, al mismo tiempo que controlan las interacciones que se llevan a cabo. Este tipo de *software* se denomina *Learning Design Player*, y este Proyecto Fin de Carrera está basado en el desarrollado por la OUNL, CopperCore, que es una aplicación *open source*.

1.2 PLANTEAMIENTO DEL PROBLEMA

Este Proyecto Fin de Carrera surge con la idea de mejorar un aspecto básico en el reproductor de *Learning Design* de CopperCore, como es la introducción de adaptaciones en tiempo de ejecución. Es necesario ofrecer a los instructores del proceso de enseñanza la posibilidad de modificar el diseño inicial para hacer frente a los posibles eventos que se producen durante el desarrollo del proceso de aprendizaje. Estos eventos quedan reflejados a través de los siguientes ejemplos:

- Imaginemos que los estudiantes han elegido el tema en el que trabajar y se encuentran inmersos en la realización de una serie de actividades. Supongamos que uno de los estudiantes implicados en la actividad llega tarde. Comienza su trabajo, y mientras él está en la primera actividad, los demás alumnos se encuentran en la última. Entonces el profesor decide completar las actividades de este alumno para que se iguale con los demás.

- Supongamos la situación en que el proceso de aprendizaje ha comenzado. Los alumnos están realizando una actividad, que consiste en responder a las cuestiones de un test. El profesor se percata de que una de las cuestiones está mal redactada, y decide cambiarla.

- Supongamos que los estudiantes se encuentran inmersos en la parte de discusión de una actividad. El profesor decide ofrecer a sus alumnos un nuevo material disponible, consistente en parte de una conferencia dada por un experto en el tema discutido, ya que piensa que será de gran interés para ellos.

Tal y como se ha expresado en el apartado anterior, CopperCore es un *software* desarrollado para dar soporte a los procesos de aprendizaje, si bien presenta algunas carencias. Como solución a este problema, el profesor de la Universidad Carlos III de Madrid Telmo Agustín Zarraonandia Ayo propuso en su tesis doctoral “Adaptaciones de unidades de aprendizaje en tiempo de ejecución” [18] un modelo de adaptación que permitiese la extensión de IMS *Learning Design* y posibilitase la descripción de modificaciones tanto sobre la definición de los diseños de proceso de aprendizaje

desarrollados como sobre su comportamiento en ejecución. Además, incluía un mecanismo para la implantación del modelo que permitiese mantener la lógica de las adaptaciones separada de la información original de la UOL de tal forma que pudiesen definirse y aplicarse en tiempo de ejecución adaptaciones no previstas con anterioridad al comienzo de la misma.

Para la implementación de este mecanismo se ha partido de la definición de *Adaptation Poke*, que son descripciones, en lenguaje XML, de pequeñas adaptaciones sobre algunos elementos de un proceso de aprendizaje. Un *Adaptation Poke* describe la acción a realizar (actualización, adición o eliminación) e incluye tanto la información referente a los nuevos elementos a añadir como los nuevos recursos a utilizar. Tras el procesamiento de un *Adaptation Poke* la UOL quedará en estado consistente.

Estos *Adaptation Poke* serán analizados, interpretados y ejecutados en el motor de ejecución CopperCore, por lo que, para poder realizar la implementación, se ha partido del código fuente versión 3.2 disponible en *sourceforge*. Es por tanto preciso reseñar que no se parte de cero, sino que se hace uso de un código ya existente, prácticamente indocumentado y de gran tamaño.

1.3 OBJETIVOS DEL PROYECTO

El principal objetivo de este Proyecto Fin de Carrera es el de desarrollar un mecanismo capaz de conseguir que los procesos de aprendizaje ejecutados en el motor de ejecución CopperCore, especificados mediante el lenguaje de modelado educativo IMS *Learning Design*, puedan ser adaptados en tiempo de ejecución. El módulo desarrollado será capaz de interpretar estas adaptaciones y de aplicarlas, tal y como se describe en el modelo de adaptación y en el mecanismo de implementación propuestos en la tesis doctoral “Adaptaciones de unidades de aprendizaje en tiempo de ejecución” [18], realizada por Telmo Agustín Zarraonandia Ayo, profesor en la Universidad Carlos III de Madrid.

Para ello, el módulo de adaptación será desarrollado cumpliendo las siguientes características:

- Se partirá del código fuente versión 3.2 de CopperCore.
- Será implementado siguiendo la tecnología J2EE.
- Será capaz de leer un *Adaptation Poke*, escrito en XML.
- Será capaz de ejecutar las acciones indicadas en un *Adaptation Poke*, y dejar la UOL en estado consistente.

A continuación se relatan una serie de objetivos específicos que debe cumplir el mecanismo a desarrollar:

- Modificación del estado del proceso. El tutor deberá ser capaz de controlar el estado de las distintas actividades del proceso, dándolas por concluidas o iniciándolas, según le convenga.

- Modificación de los recursos empleados. El tutor será capaz de cambiar los recursos asociados a cualquier actividad.

- Modificación del diseño del proceso. El tutor será capaz de rediseñar el contenido del proceso, añadiendo, modificando o eliminando actividades o *environments*.

1.4 ESTRUCTURA DEL DOCUMENTO

En esta sección se encuentra descrito cada capítulo del presente documento.

- Capítulo 1: Introducción. Se indica el contexto de este Proyecto Fin de Carrera, así como el planteamiento del problema, los objetivos perseguidos y la estructura del documento.

- Capítulo 2: Metodología de trabajo y planificación. Se expone y define la metodología seguida por este Proyecto Fin de Carrera y la planificación llevada a cabo, así como los costes generados.

- Capítulo 3: Estado del Arte. Se profundiza en el contexto en el que se desarrolla este Proyecto Fin de Carrera.

- Capítulo 4: Análisis. Se especifican los requisitos y se detallan los casos de uso.

- Capítulo 5: Diseño. Se detalla la solución llevada a cabo, explicando el funcionamiento interno de CopperCore, necesario para entender el mecanismo desarrollado, y la relación entre los diferentes componentes de la arquitectura del sistema.

- Capítulo 6: Implementación. Se explica cómo se ha codificado la solución.

- Capítulo 7: Manual. Se describen los pasos necesarios para configurar el entorno y para ejecutar tanto CopperCore como el módulo de extensión desarrollado.

- Capítulo 8: Evaluación. Se detalla el plan de pruebas llevado a cabo.

- Capítulo 9: Conclusiones. Se destacan los conceptos más interesantes.

2 METODOLOGÍA DE TRABAJO Y PLANIFICACIÓN

En esta sección se expone la metodología de trabajo que se ha seguido, así como las fases del proyecto resultantes de la aplicación de la misma. También se describen las diferentes herramientas empleadas, así como los costes generados. Por último, se hace una comparación entre la planificación inicial estimada y la planificación real.

2.1 METODOLOGÍA DE TRABAJO

La ISO (*International Organization for Standarization*) define el ciclo de vida de un *software* como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto *software*, abarcando desde la definición hasta la finalización de su uso.

Más allá de esta definición, el aplicar una metodología para el desarrollo de *software* es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas probabilidades de éxito, además de obtener un producto correcto y libre de errores. Esta sistematización indica cómo dividir un proyecto en módulos más pequeños llamados etapas y las acciones que corresponden en cada una de ellas, lo que nos ayuda a definir entradas y salidas para cada una de las etapas y, sobre todo, normaliza el modo de administrar el proyecto.

Este Proyecto Fin de Carrera presenta las siguientes características:

- Planificación sencilla.
- Se comienza a trabajar sobre una aplicación ya existente.
- Se conocen los requisitos al comienzo.

Ajustándonos a las características recién mencionadas, el modelo de ciclo de vida que se va a seguir en este Proyecto Fin de Carrera es conocido como ciclo de vida en cascada, modelo clásico, modelo tradicional o modelo lineal secuencial, y puede verse en la Ilustración 2.1. Fue propuesto por Winston Royce en el año 1970. Es el más básico de todos los modelos y sirve como bloque de construcción para los demás modelos de ciclo de vida. Dice que el desarrollo de *software* se realiza a través de una secuencia simple de fases. Cada fase tiene un conjunto de metas bien definidas, y las actividades dentro de cada una de ellas contribuyen a la satisfacción de las metas de esa etapa. Después de cada etapa se realizan una o varias revisiones para comprobar si se puede pasar a la siguiente. Es un ciclo de vida que admite iteraciones.

Una de sus principales ventajas, además de su planificación sencilla, es la de proveer un producto con un elevado grado de calidad sin necesidad de un personal altamente cualificado. Por otro lado, este modelo de ciclo de vida es también adecuado para el desarrollo de productos que no son novedosos, lo que es aplicable a nuestro caso, ya que se trata de la implementación de un módulo de extensión de un *software* ya existente.

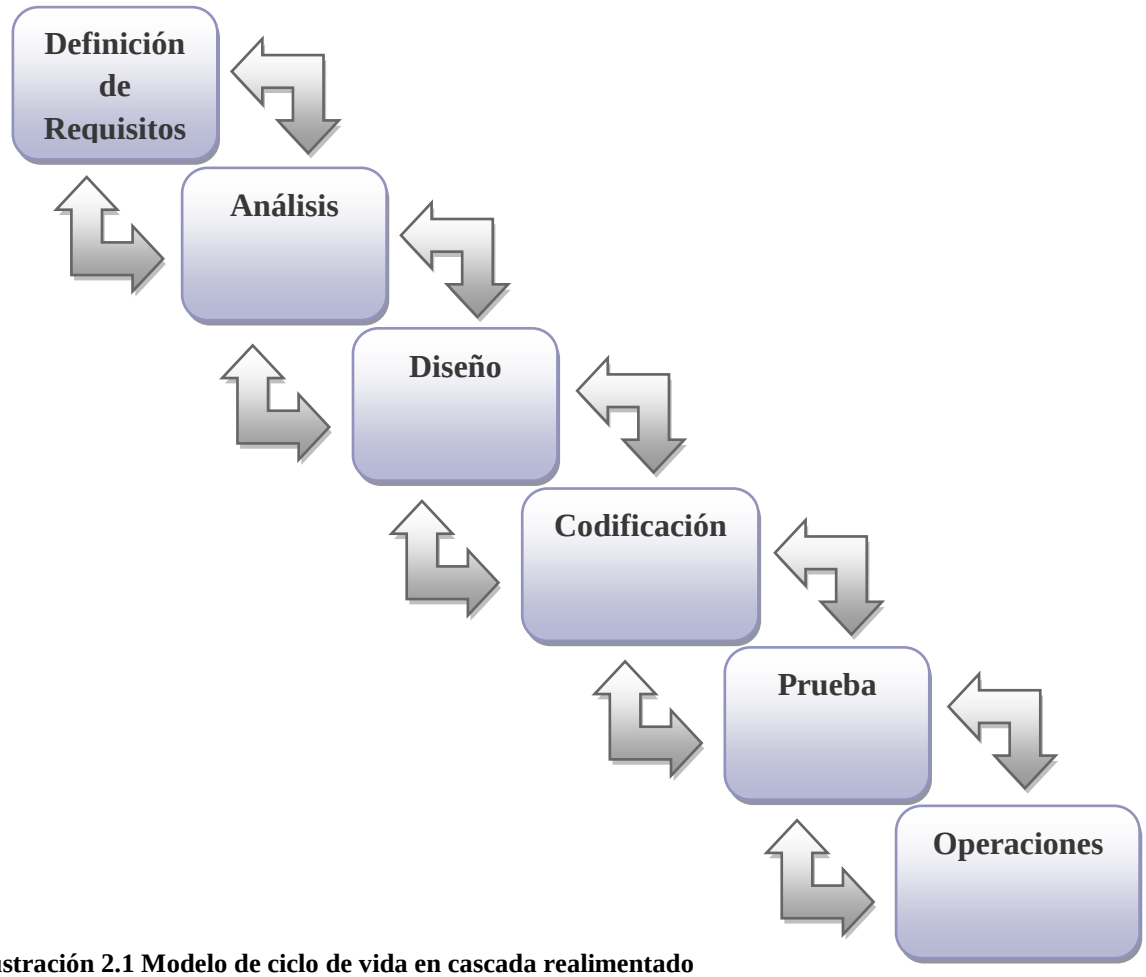


Ilustración 2.1 Modelo de ciclo de vida en cascada realimentado

Se puede considerar como un inconveniente la necesidad de contar con todos los requerimientos (o la mayoría) al comienzo del proyecto y, si se han cometido errores y no se detectan en la etapa inmediatamente siguiente, es costoso y difícil volver atrás para realizar la corrección posterior. Además, los resultados no se ven hasta que no nos encontremos en las etapas finales del ciclo, por lo que cualquier error detectado conlleva un retraso y aumenta el costo del desarrollo en función del tiempo que consuma la corrección de estos. Esto no supone un excesivo riesgo en este Proyecto Fin de Carrera, puesto que se cuenta con la mayoría de los requisitos al principio, al menos de manera general, pues se trata de un módulo de mejora de un producto ya existente, y las funcionalidades a desarrollar se encuentran definidas en los objetivos. Por otro lado, la detección de errores no supondría una demora excesiva, ya que se trata de un proyecto pequeño.

En este caso, un ciclo de vida lineal no es justificable, puesto que no permite volver atrás una vez finalizada una etapa, lo que implicaría la no corrección de los posibles errores detectados. Por otra parte, un modelo de gestión más elaborado tampoco se justifica, ya que se trata de un proyecto de pequeñas dimensiones.

2.2 FASES DEL PROYECTO

A raíz de la metodología de trabajo expuesta con anterioridad, se establecieron las siguientes fases o tareas para el desarrollo del presente Proyecto Fin de Carrera:

- Estudio preliminar. Esta fase se inicia con la elección del Proyecto Fin de Carrera y con las primeras reuniones con el tutor, en las que se discutirá sobre el problema que nos atañe. Se pueden distinguir tres partes:

  - ✓ Planteamiento del problema. Se detecta cuál es el problema que genera la necesidad de un desarrollo *software*.

  - ✓ Esbozo de la solución. Se define preliminarmente cómo se va a resolver el problema.

  - ✓ Planificación del proyecto. Tras la determinación del problema y su posible solución, es necesario definir la mejor manera de gestionar tanto los recursos disponibles como el tiempo necesario para su implementación. En esta fase se detallarán las tareas en que se dividirá el proyecto y los costes asociados a cada una de ellas.

  - ✓ Estudio del estándar IMS LD. En esta fase se asimila el estándar IMS LD, al que da soporte CopperCore.

  - ✓ Estudio del código fuente. Esta tarea es la que más esfuerzo requiere, pues consiste en comprender el código de CopperCore (más de 370 clases), que apenas contiene documentación de apoyo.

- Análisis del sistema. Una vez realizado el estudio preliminar, se procede a llevar a cabo el análisis del sistema a implementar. Esta fase comprende las siguientes tareas:

  - ✓ Estudio de la situación actual. Se profundizará en el conocimiento del contexto en el que se desarrolla el proyecto.

  - ✓ Estudio de las tecnologías. Se realizará un estudio sobre las tecnologías involucradas en el proyecto (Java, XML,…).

  - ✓ Definición de requisitos. Se determinan y especifican los requisitos que debe satisfacer la aplicación.

  - ✓ Definición de casos de uso. Se especifican una serie de casos de uso que reflejarán cada una de las funcionalidades a desarrollar.

- Diseño del sistema. Una vez realizado el análisis, se procede al diseño de una solución que satisfaga los requisitos recogidos. Se divide en:

  - ✓ Diseño de la arquitectura. Se detallan las relaciones entre los distintos componentes del sistema.

  - ✓ Diseño detallado. Se especifican los diferentes componentes.

- Codificación. Una vez concluido el diseño de la solución se procederá a su implementación. Para ello se partirá de un código fuente (CopperCore, versión 3.2), en el que se realizarán una serie de modificaciones. Se implementarán las soluciones de cada uno de los objetivos propuestos, estableciéndose diversas reuniones con el tutor para encontrar arreglo a los diferentes problemas encontrados, la mayoría de ellos debido a que se trata de la implementación de un estándar, en el que se ofrecen recomendaciones a seguir, por lo que las soluciones desarrolladas son libres. Este desarrollo se llevará a cabo con la tecnología J2EE, al estar CopperCore diseñado para ella. Esta es sin duda la fase más extensa del proyecto, debido a la gran cantidad de problemas encontrados para poder conseguir una aplicación que se ajuste a los objetivos planteados. Se divide en dos partes:

  - ✓ Codificación de los parseadores. Necesario para el tratamiento de los ficheros XML.

  - ✓ Codificación de las funcionalidades. Se completa el código con la implementación de todas las funcionalidades necesarias.

- Evaluación. Una vez concluida la implementación de la solución del problema se establecerán y ejecutarán una serie de pruebas experimentales sobre la aplicación con el fin de demostrar tanto la consecución de los objetivos planteados como la utilidad, factibilidad y calidad de la solución propuesta. Las pruebas se plantean de tal forma que se exploren todos los posibles caminos del código, de manera que se cumpla con la calidad deseada. Esta fase se puede dividir en las siguientes tareas:

  - ✓ Definición de las pruebas. Comprueban que los diferentes componentes de la aplicación funcionan de manera separada, facilitando una matriz de trazabilidad entre los diferentes casos de prueba y los requisitos del sistema.

  - ✓ Ejecución de las pruebas. Se detalla el resultado obtenido con cada una de las pruebas.

- Generación de la documentación de la memoria. Esta fase comprende el proceso de elaboración de este documento. Se establecen los objetivos perseguidos, se define el contexto del problema y se lleva a cabo un estudio del estado del arte, en el que se analizan diferentes aspectos de los procesos de aprendizaje. Posteriormente se pasa al análisis y a la definición del problema, en la que se analizan las principales carencias y problemáticas que pueden surgir a la hora de implementar mediante un EML un proceso de aprendizaje asistido por ordenador y supervisado. Más adelante, se describe la implementación desarrollada y se define el método de evaluación seguido. Por último, se exponen las conclusiones extraídas. Esta fase se ha desarrollado conjuntamente con las demás tareas mencionadas, pues de esta forma se tienen más claros y recientes tanto los conceptos como las decisiones tomadas, además de hacer más llevadero todo el proceso.

En la Ilustración 2.2 se puede ver la metodología de trabajo empleada, en la que se indican las diferentes tareas llevadas a cabo.

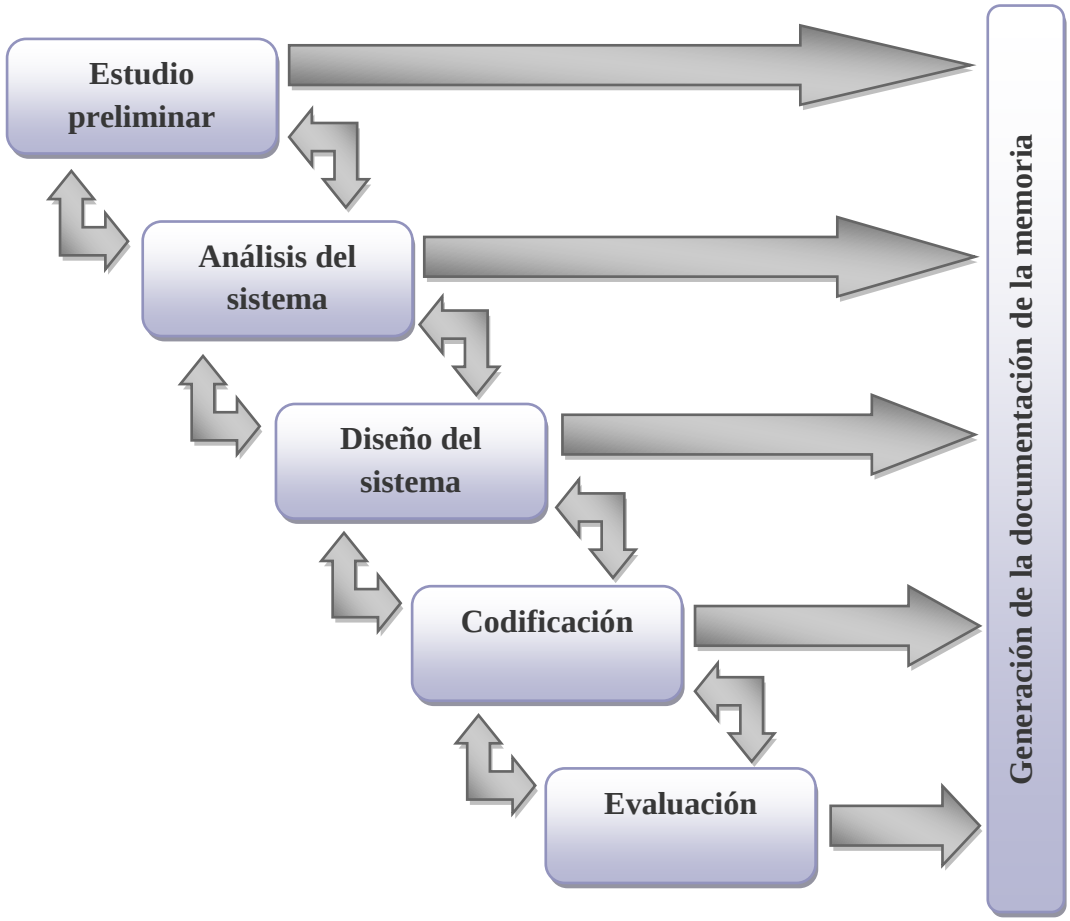


Ilustración 2.2 Fases del proyecto

2.3 PLANIFICACIÓN

El que se defina una metodología de trabajo y se planifiquen las diferentes tareas a realizar no implica el éxito de ningún proyecto, ni tampoco que se cumplan las fechas de entrega propuestas, pues siempre pueden surgir problemas y resultar que tareas que a priori parecen sencillas, no lo sean en realidad. La planificación llevada a cabo se estableció en el momento del estudio preliminar del problema, si bien la fecha de entrega se ha ido retrasando poco a poco debido a los problemas surgidos. A continuación se va a detallar tanto la planificación inicial (Ilustración 2.3) como la final (Ilustración 2.4), para poder comparar la diferencia existente entre una planificación estimada inicialmente y otra real, analizando las tareas que han llevado más tiempo del previsto y cuáles menos.

Para empezar, hay que indicar que la duración total del proyecto ha sido de seis meses, desde el 1 de septiembre de 2009 hasta el 9 de febrero de 2010, en total 116 días laborables. La intención inicial era que la fecha de entrega fuera el 11 de diciembre de 2009, con una duración aproximada de 74 días. En la Tabla 2.1 se puede ver una comparación entre las dos planificaciones, donde el total acumulado de días contiene la suma de la dedicación en días de cada fase.

Fase / Mes	Duración real (días)	Duración estimada (días)
Estudio preliminar	33	8
Análisis del sistema	8	8
Diseño del sistema	9	7
Codificación	26	25
Evaluación	12	5
Memoria	32	25
TOTAL ACUMULADO DÍAS	120	78
TOTAL REAL DÍAS	**116**	**74**

Tabla 2.1 Comparación entre días estimados y días reales

El principal factor que ha llevado a acumular una desviación de 42 días laborables (116 frente a 74) ha sido el estudio preliminar, con un desvío de 25 días. Esto se debe a que se ha partido de un código fuente ya existente para la implementación del sistema, el cual, además de ser muy extenso (más de 370 clases), apenas estaba documentado. Esto ha supuesto diversos problemas a la hora de entender el funcionamiento interno de CopperCore. Además, el estudio del estándar IMS LD también ha supuesto un esfuerzo extra.

En las fases de análisis, diseño y codificación el desvío ha sido de apenas 3 días, por lo que se deduce que la planificación fue buena. Sin embargo, en la fase de evaluación se ha producido una demora de 7 días, debido a que diversas pruebas han mostrado problemas que no se habían tenido en cuenta. Por último, el desvío de tiempo en la redacción de la memoria (7 días) se ha debido a la costumbre de generar documentos en grupo, y esta vez se ha tenido que afrontar el trabajo de manera individual.

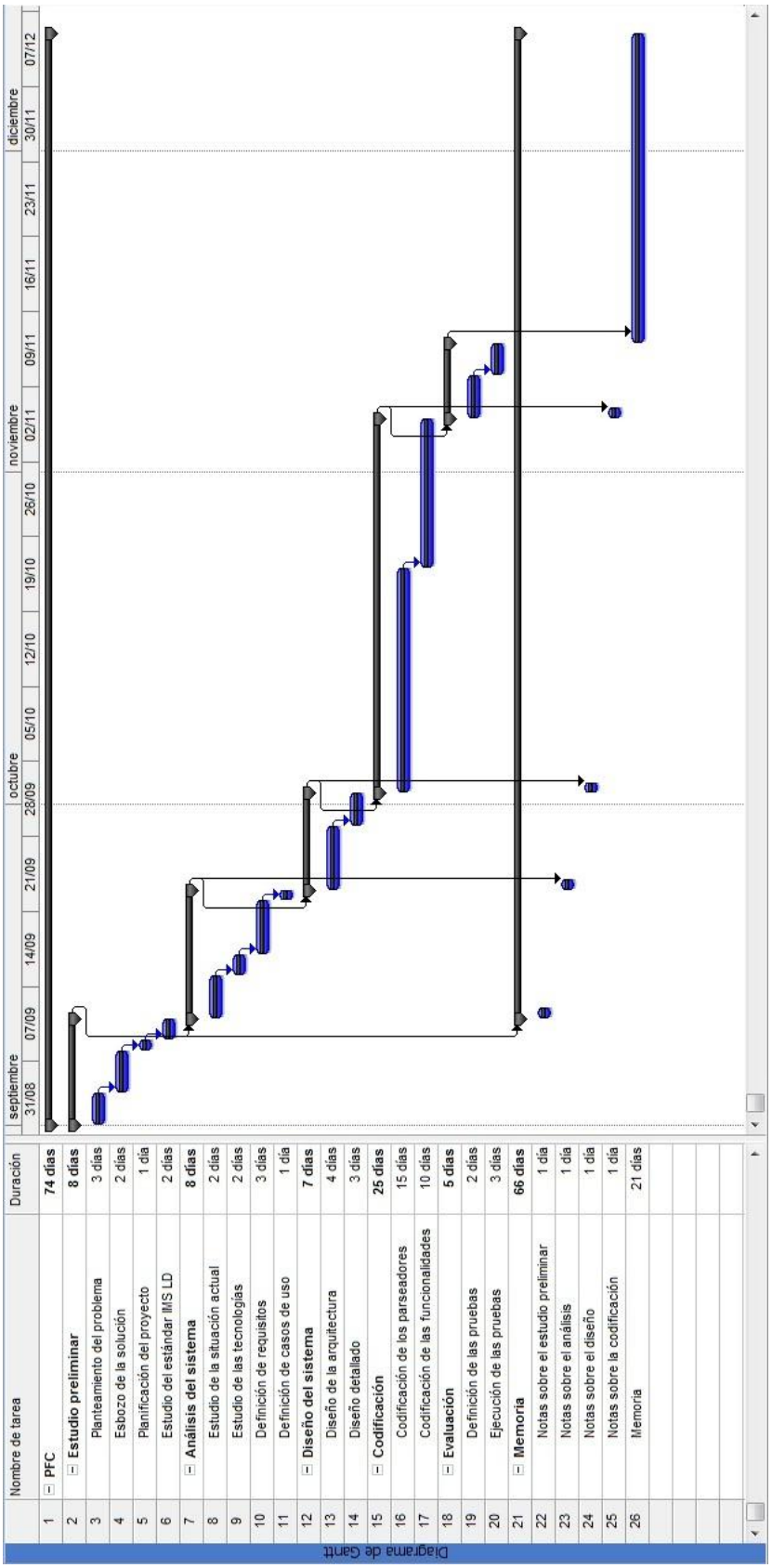


Ilustración 2.3 Diagrama de Gantt con la planificación inicial

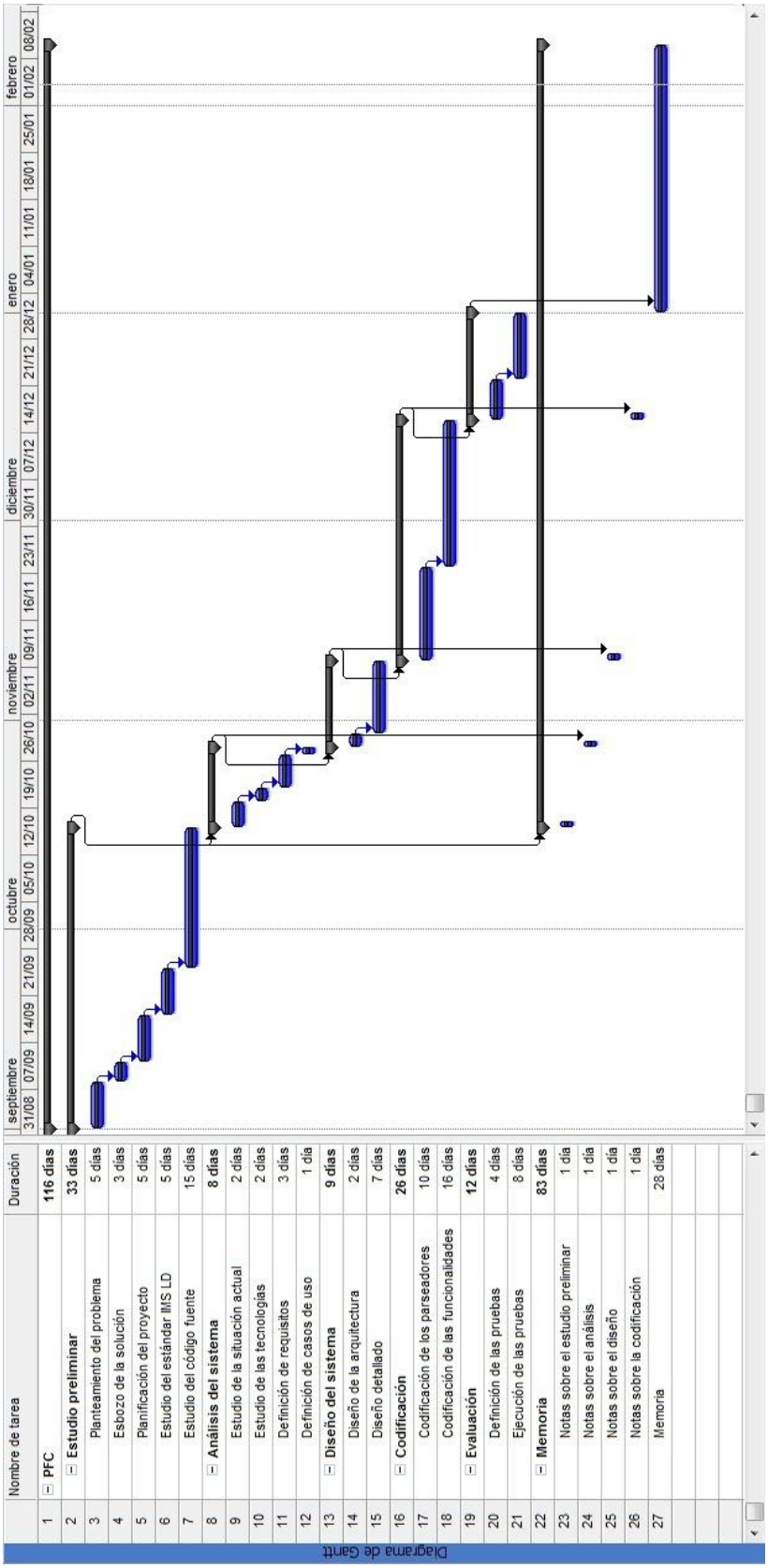


Ilustración 2.4 Diagrama de Gantt con la planificación real

2.4 HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS

Para la elaboración de este proyecto se van a utilizar diversas herramientas y tecnologías, tanto para las fases de análisis, diseño, codificación y evaluación como para la elaboración del presente documento.

2.4.1 J2EE

CopperCore es un motor de ejecución para IMS *Learning Design* que opera bajo la tecnología J2EE, que ha sido desarrollada por Sun Microsystems [19].

La plataforma Java 2 Edición Empresarial (J2EE, *Java 2 Enterprise Edition*) define el estándar para desarrollar múltiples aplicaciones de empresa. La plataforma J2EE simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proveyendo de un completo conjunto de servicios para esos componentes y manejando una gran cantidad de detalles sobre el comportamiento interno de las aplicaciones automáticamente, sin añadir mayor complejidad a la programación.

La plataforma J2EE obtiene los beneficios de muchas de las características de la plataforma J2SE, tales como la portabilidad, el API JDBC para acceso a la base de datos, la tecnología CORBA para la interacción con los recursos existentes de la empresa y un modelo de seguridad que protege los datos incluso en las aplicaciones de Internet. Construido sobre esta base, la plataforma J2EE proporciona un soporte total para los componentes *Entreprise JavaBeans*, *Java Servlets* API, *JavaServer Pages* y la tecnología XML.

El estándar J2EE incluye especificaciones completas y test de conformidad para asegurar la portabilidad a través de la amplia variedad existente de sistemas empresariales capaces de soportar la plataforma J2EE.

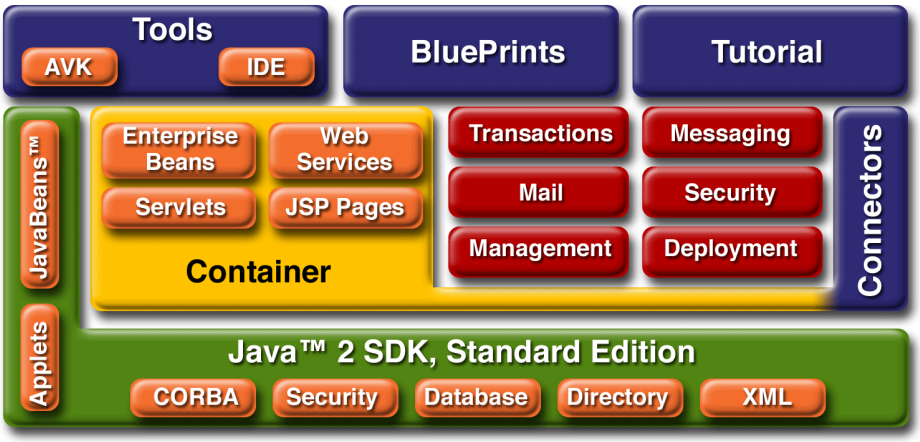


Ilustración 2.5 Servicios ofrecidos por J2EE

El éxito de la plataforma J2EE llega a ser tal en las empresas que más de las dos terceras partes de los administradores de desarrollo de *software* usan esta plataforma para desarrollar y ejecutar sus aplicaciones.

Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados. Todo ello conlleva a que los desarrolladores puedan concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

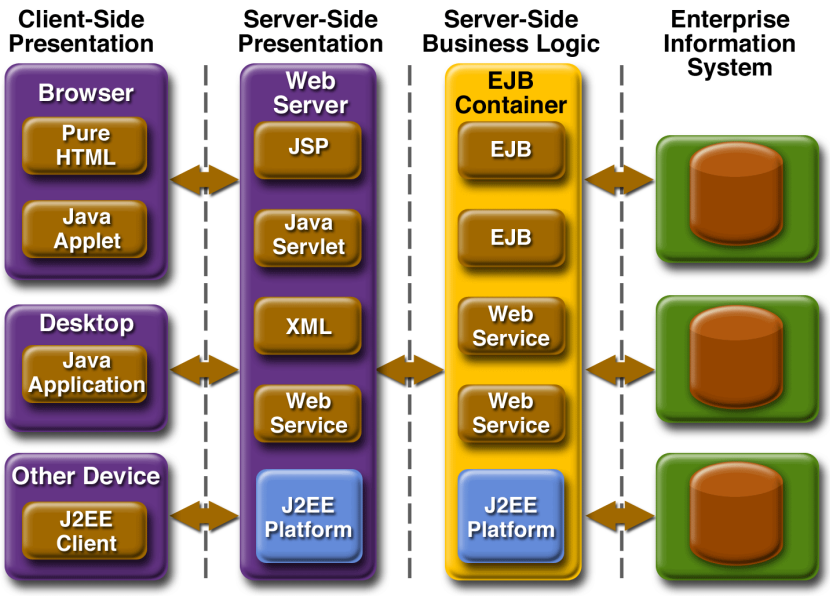


Ilustración 2.6 Modelo de Aplicación Empresarial de J2EE

2.4.2 ANT

Para poder realizar la compilación del código fuente de CopperCore es necesario utilizar Apache Ant [20].

Apache Ant es una herramienta para compilar basada en Java que tiene la ventaja de no depender de las órdenes del intérprete de comandos de cada sistema operativo, sino que se basa en archivos de configuración XML y en clases Java para la realización de las distintas tareas, siendo idónea como solución multiplataforma. Esta herramienta fue creada por James Duncan Davidson.

Para la utilización de Ant es necesario disponer de una distribución binaria de Ant y tener instalada la versión 1.4 o superior del JDK.

Una de las desventajas de esta herramienta es que al estar basada en XML, los archivos Ant deben ser escritos en XML. Esto es no sólo una barrera para los nuevos usuarios, sino también un problema en los proyectos muy grandes, cuando se construyen archivos muy grandes y complejos.

2.4.3 ECLIPSE

A la hora de desarrollar el código de la aplicación se ha usado un entorno de programación, en este caso Eclipse [21], que además de permitir ver todos los módulos que conforman el código fuente, ofrece una serie de utilidades que facilitan la codificación, como puede ser la navegación por las librerías de que dispone, la navegación por los propios componentes del código, etc.

Eclipse es una plataforma de desarrollo abierta compuesta por *frameworks* extensibles, herramientas y entornos de ejecución para construir, desarrollar y administrar *software* a lo largo de todo su ciclo de vida. Fue originalmente creado por IBM en el año 2001.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos para proporcionar toda su funcionalidad al frente de la plataforma del cliente, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de *software*. Además de permitir a Eclipse extenderse usando otros lenguajes de programación como C/C++ y Phyton, permite trabajar con lenguajes para procesado de texto como Latex, aplicaciones en red como Telnet y Sistema de Gestión de Bases de Datos. Ofrece soporte para Java y CVS en el SDK de Eclipse.

2.4.4 JDK 1.4

La aplicación que se va a desarrollar va a estar programada en Java, más concretamente en la versión 1.4.2 proporcionada por JDK.

Java es ante todo un lenguaje de programación moderno, diseñado en la década de los noventa, y proporciona una potencia, una robustez y una seguridad que muy pocos lenguajes pueden igualar, sin olvidar su rasgo más conocido: es totalmente portable.

Si bien Java tiene sus bases en lenguajes como C y C++, los supera con creces y sería un error pensar que es una simple evolución de estos. Java tiene entidad propia y características novedosas y potentes que hacen de él no sólo una apuesta de futuro, sino también un lenguaje con un presente claro y asentado. No obstante, toda la potencia que Java proporciona tiene un coste; es necesario asimilar muchos conceptos, técnicas y herramientas que en muchos casos son totalmente nuevas y hacen que la curva de aprendizaje sea pronunciada. Sin embargo, una vez superados los primeros escollos, los resultados son espectaculares.

2.4.5 XML

El lenguaje XML está presente en este proyecto tanto en los archivos necesarios para compilar el código fuente como en el modelo desarrollado para alcanzar la solución planteada al problema inicial, ya que los archivos en los que se encuentran las modificaciones que se quieren llevar a cabo sobre una UOL, al igual que estas, están especificados en este lenguaje.

XML es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos.

XML no sólo ha nacido para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Además es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad, ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Entre las principales ventajas de XML cabe destacar que es extensible, pues después de diseñado y puesto en producción es posible la adición de nuevas etiquetas.

2.4.6 MICROSOFT OFFICE 2007

Para la redacción de esta memoria se han utilizado varios de los productos incluidos dentro del paquete Microsoft Office 2007.

- Microsoft Office Word 2007. Se trata de un *software* destinado al procesamiento de textos. Es la aplicación que se ha utilizado para redactar este documento.

- Microsoft Office Excel 2007. Se trata de un *software* para manejar hojas de cálculo. Es la aplicación que se ha utilizado para realizar las tablas que contienen las pruebas unitarias realizadas, así como la matriz de trazabilidad.

- Microsoft Office Project Professional 2007. Se trata de un *software* usado para la gestión de proyectos, más concretamente para el desarrollo de planes, la asignación de recursos a tareas, dar seguimiento al proceso, administrar el presupuesto y analizar las cargas de trabajo. En este proyecto se ha utilizado para llevar a cabo toda la planificación expuesta con anterioridad.

- Microsoft Office Visio Professional 2007. Se trata de un *software* de dibujo vectorial. Se ha utilizado para la generación de gráficos UML (diagramas de casos de uso, diagramas de clase, diagramas de secuencia, etc.).

2.5 COSTES GENERADOS

En esta sección se van detallar los costes que supondrá la realización de este proyecto, desglosándolos en recursos humanos, equipamiento, fungibles, etc. La estimación de costes se ha llevado a cabo teniendo en cuenta tanto las fases en que se divide el proyecto, detalladas con anterioridad, como el tiempo de duración del mismo: 6 meses.

En primer lugar, se va a proceder con el desglose de gastos referente al personal involucrado en la realización de este proyecto, compuesto por dos personas. Los roles interpretados han sido:

- Telmo Agustín Zarraonandia Ayo: ha interpretado el rol de cliente.
- Pedro Santos Moreno: ha interpretado los roles de jefe de proyecto, analista y programador.

Esto implica que tanto el jefe de proyecto como el analista como el programador pueden trabajar en paralelo, es decir, que un día de trabajo de cada uno puede haberse realizado en la misma fecha.

En la Tabla 2.2 se muestra el precio por hora estimado para cada rol.

Rol / Precio	€ / hora
Jefe de proyecto	45
Analista	30
Programador	18

Tabla 2.2 Precio por hora de cada trabajador

Una vez estimado el coste por hora que supone cada trabajador, hay que hacer una relación entre cada fase, los días trabajados en cada una de ellas y los meses de duración del proyecto. Esto se muestra en la Tabla 2.3.

Fase / Mes	09/09	10/09	11/09	12/09	1/10	2/10
Estudio preliminar	22	11				
Análisis del sistema		8				
Diseño del sistema		3	6			
Codificación			15	11		
Evaluación				12		
Memoria		2	1	1	21	7
TOTAL DÍAS	22	24	22	24	21	7

Tabla 2.3 Días trabajados de cada mes en cada fase

Los días expresados en el total de cada mes no se corresponden con los días reales de cada mes (30, 31, 30, 31, 31 y 28 respectivamente), ya que se cuentan sólo días laborales (se suprimen sábados y domingos). Una vez establecidos los días de trabajo totales, hay que indicar qué días ha colaborado cada trabajador. Esto se puede apreciar en la Tabla 2.4.

Mes / Rol	Jefe de proyecto	Analista	Programador	TOTAL DÍAS
Septiembre	1	22		23
Octubre	1	22		23
Noviembre	2	7	15	24
Diciembre	2	1	23	26
Enero	1	21		22
Febrero	1	7		8
TOTAL DÍAS	8	80	38	**126**

Tabla 2.4 Días empleados por rol y mes

Como se puede observar, el total de días laborables empleados en los 6 meses (116, Tabla 2.1 e Ilustración 2.4) no coincide con el total de días acumulado con la suma del trabajo de cada rol (126). Esto es debido a que los roles de jefe de proyecto, analista y programador los asume una misma persona, de forma que se puede desarrollar el trabajo en paralelo. Es decir, en un mismo día se pueden ejercer varios roles a la vez. Para realizar el cálculo de costes habrá que utilizar los 126 días, pues esta cifra incluye los días empleados por cada trabajador.

La siguiente estimación hace referencia al número de horas empleadas por cada día trabajado. Se va a suponer una media jornada para el analista y el programador (4 horas), mientras que para el jefe de proyecto algo menos (2 horas). Estos datos se recogen en la Tabla 2.5.

Rol / Dedicación	Días	Horas / día	TOTAL
Jefe de proyecto	8	2	16
Analista	80	4	320
Programador	38	4	152
TOTAL HORAS			**488**

Tabla 2.5 Horas totales trabajadas por rol

Con la Tabla 2.2, que incluye el precio por hora de cada trabajador, y la Tabla 2.5 se puede calcular el coste total referente al personal empleado (Tabla 2.6).

Rol / Coste	€ / h	Horas	Coste (€)
Jefe de proyecto	45	16	720
Analista	30	320	9600
Programador	18	152	2736
COSTE TOTAL (€)			**13056**

Tabla 2.6 Coste total del personal implicado en el proyecto

Una vez estimados los costes de personal, se va a proceder a analizar los cálculos de los gastos de equipamiento necesarios para la realización del proyecto. Este equipamiento incluye elementos *hardware* y *software*, además de otros recursos extra.

En cuanto a los gastos en equipamiento *hardware*, estos incluyen el uso de un portátil, una impresora para generar la documentación y un *router wifi* para la conexión a Internet. Se calculará una amortización teniendo en cuenta la duración de un equipo en 3 años y un uso de 6 meses. Se puede ver la relación entre estos datos en la Tabla 2.7.

Equipamiento / Coste	Unidades	Precio (€)	Amortización (meses)	Uso (meses)	Coste (€)
Ordenador portátil	1	1000	36	6	166,67
Impresora a color	1	150	36	6	25
Router wifi	1	60	36	6	10
COSTE TOTAL (€)					**201,67**

Tabla 2.7 Coste del equipamiento hardware

En lo que se refiere al cálculo de los gastos incurridos en equipamiento *software*, se basa en la compra de licencias para los diferentes programas informáticos usados, que se encuentran detallados en secciones anteriores. Si bien el sistema operativo usado ha sido Windows 7, este ya se encontraba instalado en el equipo, por lo que su precio ya se ha incluido en el equipamiento *hardware*. Los programas utilizados por los que hay que pagar licencia son Microsoft Office Word, Microsoft Office Excel, Microsoft Project y Microsoft Visio. También se calculará una amortización teniendo en cuenta la duración de los programas en 3 años y un uso de 5 meses (Tabla 2.8).

Licencia / Coste	Unidades	Precio (€)	Amortización (meses)	Uso (meses)	Coste (€)
Microsoft Office Word 2007	2	289	36	6	96,33
Microsoft Office Excel 2007	1	289	36	6	48,17
Microsoft Project 2007	1	779	36	6	129,83
Microsoft Visio 2007	1	450	36	6	75
COSTE TOTAL (€)					**349,33**

Tabla 2.8 Coste del equipamiento software

Por último, hay que tener en cuenta el resto de recursos fungibles, como puede ser material de oficina, cartuchos de tinta, acceso a Internet (calculado a 30 €/mes), etc. Todos estos gastos son calculados para el total de los 6 meses. Se puede observar en la Tabla 2.9.

Recurso / Coste	Coste (€)
Acceso a Internet	180
Tinta impresora	140
Material oficina	40
COSTE TOTAL (€)	**360**

Tabla 2.9 Coste de los recursos fungibles

A raíz de todos los cálculos anteriores (coste total incluido en la Tabla 2.6, Tabla 2.7, Tabla 2.8 y Tabla 2.9), tenemos que el total de gastos, sin incluir el IVA, es de (ver Tabla 2.10):

Concepto / Coste	Coste (€)
Gastos de personal	13056
Gastos de hardware	201,67
Gastos de software	349,33
Gastos fungibles	360
COSTE TOTAL (€)	**13967**

Tabla 2.10 Gasto total sin IVA

Al gasto anterior le tenemos que añadir el correspondiente 16% de IVA (ver Tabla 2.11).

Concepto / Coste	Coste (€)
Subtotal	13967
IVA (16%)	2234,72
COSTE TOTAL (€)	**16201,72**

Tabla 2.11 Coste total del Proyecto Fin de Carrera

El coste total de la realización de este Proyecto Fin de Carrera es de **dieciséis mil doscientos un euro y setenta y dos céntimos** (**16201,72 €**).

3 ESTADO DEL ARTE

En este apartado se expone la información necesaria para que el lector entienda el contexto en el que se desarrolla el proyecto. Se trata de un análisis de la importancia de la adaptación del diseño de los procesos de aprendizaje, así como las características de las mismas. Posteriormente se analizarán los dos principales enfoques posibles a la hora de dotar de soporte computacional a un proceso de aprendizaje supervisado por un instructor, así como de algunas tecnologías relacionadas. También se dará una visión de los estándares existentes y sus beneficios, comentando también el estándar específico en el que se desenvuelve este trabajo (IMS LD) y su implementación (*CopperCore*). Por último se hará una reseña al modelo de adaptación que se va a implementar en este Proyecto Fin de Carrera, extraído de la tesis doctoral del profesor de la Universidad Carlos III de Madrid Telmo Agustín Zarraonandia Ayo.

Las Tecnologías de la Información y Comunicación (en adelante TIC) se hallan hoy en día presentes en prácticamente todos los ámbitos de la vida cotidiana. Su integración dentro de los procesos de enseñanza-aprendizaje es, por tanto, una necesidad: por una parte, es necesario formar al alumno para que adquiera competencias básicas en su uso y, por otra, es deseable aprovechar las ventajas derivadas de su empleo como soporte del proceso educativo. Entre algunas de dichas ventajas cabe destacar:

- Uso de material multimedia.
- Individualización del proceso.
- Adaptación y control del proceso en manos del alumno.
- Acceso a un gran volumen de información proporcionado por Internet.
- Liberar los procesos de aprendizaje de restricciones de espacio y tiempo.

Quizás, de entre todas las ventajas mencionadas, las más perseguidas hayan sido las de conseguir procesos de aprendizaje que se adapten a las características del alumno, que puedan ser repetidos un número indeterminado de veces para distintos participantes y que puedan ser ejecutados con independencia de la ubicación física del alumno. En cualquier caso, el uso de una nueva herramienta provoca cambios dentro del nuevo proceso, creando nuevas posibilidades y enriqueciéndolo. Así pues, si bien el uso de nuevas tecnologías puede proporcionar ventajas estratégicas importantes a las instituciones educativas de todos los niveles, su utilización efectiva implica un replanteamiento o un rediseño a fondo, no sólo de los métodos de enseñanza y planes curriculares, sino también en las prácticas de trabajo y en los papeles desempeñados por profesores y alumnos.

3.1 ADAPTACIONES EN PROCESOS EDUCATIVOS SUPERVISADOS POR INSTRUCTOR

A la hora de implementar un proceso de enseñanza-aprendizaje, el instructor suele utilizar un determinado diseño instructivo que puede incluir la definición de los distintos roles de los participantes del proceso, las tareas asociadas a los mismos, el calendario del proceso, el material que va a ser empleado, evaluaciones, etc. En cualquier caso, es necesario permitir al instructor cierto grado de libertad a la hora de aplicar este plan inicial del proceso a un contexto real de ejecución. En el caso que nos ocupa, contexto puede ser cualquier información empleada para caracterizar un proceso de aprendizaje, es decir, cualquier información relevante que pueda influenciar su desarrollo. Esta definición de contexto cubre, por tanto:

- Información sobre el entorno computacional del proceso (disponibilidad de dispositivos como impresoras o escáneres, ancho de banda de red, características del ordenador, resolución del monitor, etc.).

- Información sobre las características físicas y psicológicas o requisitos de formación previa del usuario (objetivo del usuario, grado de dificultad requerido, interés del alumno, idioma, etc.)

- Información sobre las características del entorno físico actual del usuario (localización, condiciones atmosféricas, posibilidad de interacción con otros usuarios, etc.).

Al llevar a cabo el diseño del proceso de aprendizaje, los autores deben tener en mente un determinado contexto de ejecución en el que se especificará tanto el perfil del alumno como la necesaria disponibilidad de tiempo y recursos. Tratar de ejecutar el diseño en un entorno de características distintas puede requerir la introducción de adaptaciones para su adecuación, pero aún cuando sea aplicado sobre un contexto similar, la introducción de modificaciones sobre la planificación inicial puede ser necesaria. Por una parte, dos alumnos con un perfil similar pueden responder de manera distinta al desarrollo del proceso. Aspectos como la motivación, personalidad o, de manera más general, el estado mental del alumno pueden influir en su capacidad para asimilar los conceptos presentados. La interacción entre los participantes también varía de un proceso a otro, de manera que el comportamiento de unos influye en el resto modificando el resultado de las actividades. Por otra parte, la disponibilidad de los recursos y materiales didácticos también puede variar durante el propio desarrollo del proceso de aprendizaje, especialmente en aquellos procesos de cierta duración.

Así, por todo ello, los instructores suelen utilizar el diseño instructivo como punto de partida del proceso de aprendizaje, observando la evolución de los participantes durante su desarrollo e introduciendo, en caso necesario, las apropiadas modificaciones para garantizar la consecución de los objetivos pre-establecidos. La importancia de estas

modificaciones o adaptaciones radica en que, en la mayoría de los casos, el éxito del proceso depende de ellas.

3.1.1 CARACTERÍSTICAS DE LAS ADAPTACIONES

Podemos distinguir una serie de características de las adaptaciones sobre un proceso de aprendizaje soportado computacionalmente que deben ser tenidas en cuenta a la hora de implementar un mecanismo que permita su introducción:

- Predictibilidad. La necesidad de introducir la adaptación sobre la definición original del proceso de aprendizaje puede o no ser conocida previamente al comienzo del proceso.

- Momento de introducción. La adaptación se puede introducir en tres momentos distintos del ciclo de vida del proceso de enseñanza:

  - ✓ Tiempo de diseño. La adaptación se lleva a cabo modificando directamente el diseño del proceso.

  - ✓ Tiempo de publicación. La adaptación se introduce en el diseño del proceso en el momento en que este se carga en el gestor.

  - ✓ Tiempo de ejecución. La adaptación se lleva a cabo durante la propia ejecución del proceso sin que este sea interrumpido.

- Frecuencia. Una adaptación puede ser introducida de manera puntual en un momento del proceso o, por el contrario, su aplicación debe ser repetida cada cierto tiempo o cada vez que un determinado evento tiene lugar.

- Ámbito. Algunas de las adaptaciones tienen como objetivo modificar el desarrollo del proceso educativo de todos los participantes, mientras que otras únicamente deben afectar a la visión que algunos de ellos tienen del mismo. Por otra parte, una adaptación puede afectar a la definición de uno o de varios de los elementos del proceso.

- Objeto de la modificación. Ciertas aplicaciones afectan al comportamiento en ejecución del proceso, mientras que otras afectan a la definición del mismo. Así, una adaptación puede obligar a los participantes a repetir una determinada actividad, mientras que otra adaptación añade una nueva o modifica el tiempo para completar una ya existente.

- Perdurabilidad. Algunas adaptaciones modificarán el desarrollo del proceso educativo, quedando integradas de manera permanente en la definición del mismo, mientras que otras lo harán únicamente durante un determinado periodo de tiempo o durante la ejecución de una determinada instancia del proceso.

- Modificación de objetivos. Podemos distinguir tres tipos distintos de adaptaciones según cuál sea su influencia en los objetivos de aprendizaje marcados para el proceso:

  - ✓ Adaptaciones leves. Son aquellas adaptaciones que no modifican los objetivos marcados y que en principio tampoco deberían tener influencia alguna en su consecución. Por ejemplo, incrementar el tiempo asignado para completar una evaluación debido a que el recurso que debía ser empleado durante la misma no se ha encontrado disponible momentáneamente.

  - ✓ Adaptaciones moderadas. Son aquellas que modifican la manera en que se tratarán de alcanzar los objetivos de aprendizaje. Cada material educativo y cada actividad propuesta en la definición del proceso está relacionada con la consecución de uno o más objetivos de aprendizaje, de manera que cualquier modificación en la definición de los componentes del proceso podrá repercutir en la consecución de los objetivos relacionados con los mismos. De igual manera, también influirán aquellas adaptaciones que alteren la secuencia de acciones establecida en el plan inicial del proceso, como por ejemplo, forzar a los participantes a repetir cierta actividad.

  - ✓ Adaptaciones graves. Son aquellas que introducen cambios en la definición de los objetivos de aprendizaje originales del proceso. Por ejemplo, cuando debido a un reajuste en el calendario el tutor del proceso se ve obligado a eliminar parte del temario original, suprimiendo de esta forma algunos de los objetivos de aprendizaje inicialmente marcados.

- Propósito. La introducción de la adaptación obedecerá a distintos objetivos. Podemos clasificar las modificaciones en:

  - ✓ Correctivas. Aquellas adaptaciones introducidas con el fin de solucionar un determinado problema o error.

  - ✓ Adaptativas. Aquellas que persiguen ajustar el proceso a nuevas características del entorno de ejecución, del usuario, etc.

  - ✓ Perfectivas. Aquellas adaptaciones llevadas a cabo con el objeto de mejorar la calidad del proceso.

  - ✓ Evolutivas. Aquellas adaptaciones que introducen nuevas funcionalidades con el objeto de dar respuesta a nuevos requisitos.

- Evaluación. Dependiendo del propósito y de las características de la adaptación, puede ser posible estimar su grado de éxito. Así, por ejemplo, cuando una adaptación es de tipo correctiva, puede ser posible conocer si el error que dio origen a la necesidad de la adaptación ha sido subsanado y, por tanto, evaluar la acción de la adaptación. En cambio, cuando la adaptación es consecuencia de una circunstancia externa al proceso, por ejemplo si una reasignación de aulas obliga a efectuar cambios en el plan original, su evaluación no es posible.

3.2 SOFTWARE DE PROCESOS EDUCATIVOS: ENFOQUES DE DISEÑO

A la hora de analizar las soluciones informáticas que en las últimas décadas se han desarrollado con objeto de dar soporte a los procesos de enseñanza podemos distinguir dos tipos de enfoques distintos: por un lado, el seguido por los *Intelligent Tutoring Systems* (ITS) y los Sistemas de Educación Adaptativos e Inteligentes en Web (AIES), y por otro, el empleado en el desarrollo de *Computer-Based Training* (CBT) y los *Learning Management Systems* (LMS). En el primer caso el énfasis está en la obtención de sistemas capaces de adaptar automáticamente el proceso de aprendizaje a las características y necesidades del alumno, mientras que en el segundo se persigue, empleando un enfoque más comercial, el desarrollo de entornos virtuales que proporcionen los servicios necesarios para el desarrollo de experiencias educativas. Nos centraremos en este último enfoque al hallarse enmarcado dentro de él este proyecto fin de carrera.

3.2.1 SISTEMAS DE GESTIÓN DEL APRENDIZAJE (LMS)

Los *Learning Management Systems* (LMS) o Sistemas de Gestión del Aprendizaje son aplicaciones *software* que automatizan las tareas de administración, gestión y monitorización de procesos educativos. Las prestaciones y componentes de un LMS varían según la plataforma, pudiendo incluir:

- Gestión de calendario del curso.
- Gestión de los contenidos del curso.
- Elementos que faciliten la comunicación entre los participantes del curso: foros, chats, videoconferencia…
- Mecanismos para generar visiones personalizadas del curso.
- Monitorización de las actividades de los estudiantes.
- Mecanismos de evaluación.

El origen de los LMS se encuentra en los *Content Management Systems* (CMS) o Sistemas de Gestión de Contenidos, que se comenzaron a desarrollar a mediados de la

época de los noventa y que tenían por objeto facilitar la creación de forma colaborativa de documentos y contenidos, así como su gestión, publicación y presentación. La mayoría de los CMS están formados por un repositorio central donde se almacenan los contenidos y una aplicación Web que permite la gestión de los mismos, de tal forma que el aspecto del *website* podía ser actualizado por cualquier usuario autorizado desde cualquier ordenador conectado a Internet.

El uso de CMS fue rápidamente adoptado en el entorno educativo, tanto como complemento a la educación presencial, permitiendo a los usuarios acceder y recuperar los recursos empleados durante el curso, como dando soporte a la educación no presencial, sirviendo para almacenar y distribuir los mismos cursos que antes eran comercializados empaquetados en CD-ROM. Pronto se desarrollaron nuevas tecnologías con el fin de aumentar las prestaciones ofrecidas por los CMS, de tal forma que se permitiese administrar los distintos cursos y los recursos asociados a los mismos, interactuar con los alumnos, obtener informes acerca de su progreso y resultados, etc., dando lugar así al nacimiento de los primeros *Learning Management Systems* (LMS).

Dentro de este tipo de sistemas, se suelen distinguir bajo el término *Learning Content Management Systems* (LCMS) a aquellos sistemas que, proveyendo prestaciones parecidas a los LMS, se centran en la gestión de los contenidos didácticos más que en la gestión de cursos. En la práctica, la distinción entre los dos tipos de sistemas es mínima y a menudo se utilizan ambas expresiones para designar a las mismas aplicaciones. Dentro de esta categoría de sistemas, los más conocidos son WebCT y Blackboard, como entornos comerciales, y MOODLE y Claroline, como *software* libre.

De manera paralela al desarrollo de este tipo de aplicaciones surgió la necesidad de establecer estándares para el desarrollo de los contenidos educativos, de tal forma que se garantizase la operatividad de componentes entre plataformas desarrolladas por distintos fabricantes y de esa forma se preservasen las inversiones realizadas.

3.2.1.1 ESTÁNDARES

Dado el alto coste asociado a la generación de recursos educativos y la proliferación de sistemas desarrollados por distintos fabricantes, surge la necesidad de establecer acuerdos a la hora de desarrollar plataformas y contenidos de la forma que se garantice la interoperabilidad y reutilización de los recursos desarrollados.

Con este fin, diversas organizaciones han desarrollado propuestas para la estandarización de diversos aspectos del *eLearning*, entre las que cabe destacar:

- IEEE LTSC (*Learning Technology Standards Comittee of the Institute of Electrical and Electronic Engineers*) [1]. Comité formado por más de una docena de grupos de trabajo y grupos de estudio encargados de desarrollar estándares técnicos, recomendaciones y guías para la tecnología educativa.

- AICC (*Aviation Industry Computer-Based Training Comitee*) [2]. Asociación internacional que desarrolla directrices para la producción y evaluación de tecnologías relacionadas con la formación para el campo de la industria de la aviación, pionera en la estandarización de materiales para la formación profesional.

- IMS *Global Consortium Inc* [3]. Reúne a un conjunto de organizaciones comerciales, educativas y gubernamentales con el propósito de desarrollar especificaciones que sean ampliamente aceptadas y que permitan la interoperabilidad entre contenidos y entornos de aprendizaje desarrollados por distintos fabricantes.

- ADL (*Advanced Distributed Learning*) [4]. Iniciativa del Departamento de Defensa del gobierno de Estados Unidos y de la Oficina de Ciencia y Tecnología de la Casa Blanca para desarrollar principios y directrices de trabajo para el desarrollo y la implementación de formación educativa sobre tecnologías Web.

Los estándares y las especificaciones propuestos por estos organismos cubren diversos aspectos relacionados con el desarrollo de material educativo, incluyendo la manera de describir sus características y contenido, los formatos de empaquetado, la comunicación con el LMS, la descripción de evaluaciones, la descripción de perfiles de alumno, etc. A continuación se describirán aquellos cuyo uso está más extendido:

- Learning Object Metadata (LOM). Es un esquema de metadatos para la descripción de recursos educativos desarrollados por el IEEE LTSC. Este estándar define un conjunto de atributos para la precisa descripción y categorización de *Learning Object*, de tal forma que puedan ser buscados, clasificados y fácilmente recuperados según criterios a través de catálogos e inventarios, así como compartidos e intercambiados a través de distintas plataformas de aprendizaje. El modelo de datos LOM está compuesto por una jerarquía de elementos de datos, simples o agregados, que son agrupados en nueve categorías principales: *general*, *lifecycle*, *meta-metadata*, *technical*, *educational*, *rights*, *relation*, *annotation* y *classification*.

- IMS Content Package. De la necesidad de intercambiar contenidos entre diversos sistemas de aprendizaje, de autoría de contenidos y de entornos de ejecución, surge esta especificación cuyo propósito es acordar un formato para el empaquetado de recursos educativos. El elemento clave en esta especificación es el paquete, el cual está compuesto por un conjunto de ficheros físicos que pueden corresponderse con varios recursos independientes, un curso completo o varios cursos a la vez y un fichero XML denominado manifiesto, que por una parte describe los diversos contenidos y, por otra, una o más posibles organizaciones de los mismos [6]. Estas organizaciones describen de manera

jerárquica las relaciones entre los contenidos, sin especificar ningún tipo de orden de navegación o presentación de los mismos.

- IMS QTI. La especificación IMS *Question & Test Interoperability* [7] es una iniciativa del IMS que tiene como objeto proporcionar un modelo de datos que permita la representación de test, así como sus correspondientes resultados. Está compuesta por dos modelos de información distintos: por un lado, el modelo de información *Assessments Sections Items* (ASI), que especifica cómo definir los test y sus componentes y, por otro lado, el modelo *Information Results Reporting*, que especifica el modo de describir las respuestas a las posibles interacciones del usuario con los componentes de los test.

  La mínima unidad intercambiable en el modelo de información ASI es el ítem, mediante el cual se especifica una determinada pregunta y se proveen instrucciones acerca de cómo debe presentarse, cómo debe tratarse la respuesta y sobre qué información de *feedback* se debe proporcionar a la misma. Los ítems pueden ser agrupados para formar secciones y test (*assessments*) y todos ellos son susceptibles de ser apropiadamente marcados con metadatos y almacenados en bancos de objetos, así como intercambiados entre distintos LMS.

- IMS Learner Information Package. La especificación IMS *Learner Information Package* [8] está basada en la propuesta de estandarización de información de alumno *Public And Private Information* (PAPI), desarrollada por el IEEE. Su objetivo es definir una estructura de datos que permita describir, almacenar y compartir información acerca del alumno, grupo de alumnos o productos de contenido educativo entre distintos entornos de aprendizaje u otros sistemas que participen en el proceso educativo. De esta manera podemos disponer de un perfil que incluya tanto información sobre las características y preferencias del alumno como un registro del progreso y formación obtenida según vaya participando en distintas experiencias educativas.

  La información sobre el alumno se almacena en un fichero XML con su mismo nombre, el cual se encuentra dividido en once secciones distintas según el tipo de información que almacenan: *identification*, *goal*, *qcl* (*qualifications*, *certifications and licenses*), *activity*, *transcript*, *interest*, *competency*, *affiliation*, *accessibility*, *security_key* y *relationship*. El propio alumno es el responsable de definir qué parte de la información puede compartirse entre sistemas y cuál no.

- SCORM. *Shareable Content Object Reference Model* es una propuesta de ADL que integra colecciones de especificaciones y estándares desarrollados por organizaciones como IMS, AICC, ARIADNE [9] o IEEE LTSC en un único marco de referencia. Utiliza el término *Shareable Content Object* (SCO) para designar la mínima agrupación de contenidos capaz de ser ejecutada y comunicarse con un LMS. La especificación está compuesta por:

- ✓ Entorno de ejecución. Formado por un protocolo específico que seguirán los LMS en la ejecución de los SCO, un modelo de datos que especifica un conjunto mínimo de información sobre el SCO que podrá registrarse en diferentes LMS y un *Application Programming Interface* (API) que establece un conjunto de funcionalidades predefinidas que permitirán la comunicación entre el SCO y el LMS.

- ✓ Modelo de agregación de contenidos. Permite la composición, etiquetado y empaquetado del contenido educativo. Emplea la especificación IMS *Content Package* para agrupar los contenidos educativos abajo una determinada organización descrita en el fichero manifiesto.

- ✓ Modelo de secuenciación y navegación. Incluye un modelo de información derivado de la especificación IMS *Simple Sequencing* [10], que permite a los desarrolladores describir diferentes secuenciaciones del material educativo, y un modelo de navegación, que define un conjunto de eventos que podrán ser disparados por el alumno y que, a su vez, activarán las distintas secuencias de contenidos.

- IMS Common Cartridge. IMS *Common Cartridge* [11] combina tres de las especificaciones de *eLearning* más difundidas en la actualidad (IMS *Content Package*, IMS *Question & Test Interoperability* y *Learning Object Metadata*) y proporciona compatibilidad con SCORM 1.2 y SCORM 2004. Este estándar nace con el propósito de convertirse en el estándar de contenidos de mayor aceptación del mercado y cuenta para ello con el apoyo tanto de los más importantes productores de contenido (McGraw-Hill, Digital Spirit, Thomson, etc.) como de las más importantes plataformas de *eLearning* (ANGEL, Blackboard, Desire2Learn, etc.).

3.3 LENGUAJES DE MODELADO EDUCATIVO

Para describir un proceso de enseñanza-aprendizaje no es suficiente con especificar cuáles son los componentes y recursos que van a ser empleados en el mismo, sino que también es preciso describir explícitamente las actividades que serán llevadas a cabo y el uso que hará cada una de ellas de los recursos educativos, los roles de los participantes en dichas actividades, las características que el entorno del proceso debe cumplir para que esas actividades puedan ser llevadas a cabo de forma adecuada, etc. Los Lenguajes de Modelado Educativo (EML) surgen con el fin de poder llevar a cabo este tipo de descripciones cubriendo diferentes tipos de enfoques pedagógicos. En esta sección se analizarán las principales características de la especificación sobre la que se sustenta este proyecto, IMS *Learning Design*, si bien existen otras propuestas de EML como:

- PALO. Desarrollado por la Universidad Nacional de Educación a Distancia (UNED).

- OUNL-EML. Desarrollado por la *Open Universiteit Nederland*. Fue usado como punto de partida para el desarrollo del IMS *Learning Design*.

- LAMS (*Learning Activity Management System*) [12]. Inicialmente construido basándose en la especificación IMS LD, finalmente adoptó sus propias soluciones para posibilitar la creación de diseños de aprendizaje formados por conjuntos de actividades secuenciales, opcionales o paralelas.

- LDL (*Learning Design Language*) [29, 30]. Iniciativa elaborada desde un enfoque centrado en el instructor desarrollada por las Universidades de Savoie y Grenoble como una alternativa a IMS LD.

3.3.1 IMS LEARNING DESIGN

IMS *Learning Design* (IMS LD) permite la creación de descripciones completas, abstractas y portables del enfoque pedagógico a emplear a lo largo de un determinado curso, de forma que puedan ser luego puestas en práctica a través de motores de ejecución compatibles con la especificación. Está basado en el OUNL-EML y proporciona los elementos necesarios para la descripción de las actividades que desempeñarán los distintos participantes de un proceso educativo. Además, integra otras muchas especificaciones como IMS *Content Packaging* [6], IMS *Simple Sequencing* [10], *Learning Objects Meta-Data Specification* [5], etc. Cuando una descripción de un LD se incluye como organización en el fichero manifiesto de un *Content Package* obtenemos una *Unit of Learning* (UOL). Una UOL contiene toda la información requerida para llevar a cabo un proceso educativo, incluyendo tanto información pedagógica como información sobre la localización y el empleo de los recursos necesarios. Una vez definida, el programa adecuado interpretará dicha información y proveerá a los participantes del proceso con el apropiado interfaz para desarrollar las actividades propuestas.

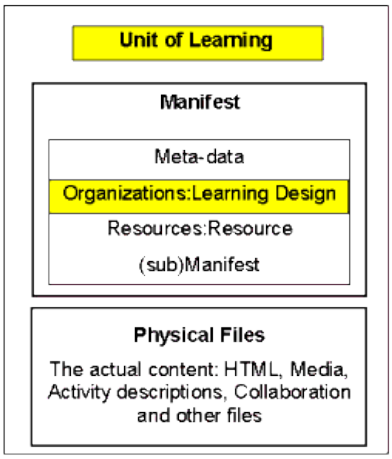


Ilustración 3.1 Estructura de una UOL

En la Ilustración 3.1 se puede ver la estructura de una UOL, compuesta mediante la inclusión de un IMS LD dentro del apartado *Organizations* de un IMS *Content Packaging*.

3.3.1.1 REQUISITOS DE DISEÑO

Algunos de los requisitos de diseño de IMS LD más relevantes son los siguientes:

- Permitir la descripción, formalización e implementación de distintos procesos de aprendizaje y de distintas aproximaciones educativas.
- Permitir la implementación de UOL consistentes en actividades heterogéneas.
- Permitir el descubrimiento y la interoperabilidad de estas UOL.
- Aprovechar las especificaciones y estándares ya existentes en los casos en que sea posible.
- Permitir la inclusión en las actividades de múltiples participantes ejerciendo distintos roles para dar soporte a experiencias de aprendizaje en grupo y colaborativas/competitivas.

3.3.1.2 ESPECIFICACIÓN DE IMS LD

Para satisfacer estos requisitos es necesario estudiar cuales son los elementos esenciales de los procesos de aprendizaje. Una vez encontrados estos elementos, se construye sobre ellos una formalización para permitir el intercambio y la interoperabilidad. Puesto que los diseños de procesos de aprendizaje no son conocidos a priori, las definiciones deben ser suficientemente abstractas para así poder ser empleadas en múltiples escenarios educativos. La abstracción sobre la que se construye la especificación IMS LD es la formada por actividades de aprendizaje (cualquier actividad en la que un participante se puede involucrar durante un proceso de aprendizaje) y flujos de aprendizaje (planteamiento de un número de actividades que deben realizarse en un determinado orden, con unos determinados participantes y, habitualmente, con varios caminos posibles en función de los resultados obtenidos por los distintos participantes).

Para la descripción de un proceso de aprendizaje IMS LD utiliza una analogía con una obra de teatro (*play*), de tal forma que un proceso de aprendizaje puede ser interpretado por distintos participantes haciendo uso de distintos recursos. Cada participante desempeñará un rol (*role*) determinado, *learner* o *staff*, dentro de una serie de actos (*acts*), llevando a cabo una serie de actividades. Dichas actividades se desarrollan empleando los *learning-objects* y/o *services* (chats, foros, buscadores, etc.) que conforman el entorno (*environment*) de cada actividad (*activity*). Un acto termina cuando todos los participantes han llevado a cabo las actividades especificadas para cada uno de sus roles o bien cuando se sobrepasa un determinado límite de tiempo. Al finalizar un acto, da comienzo el siguiente. Los *plays* agrupan a los actos y sólo

concluyen una vez que todos ellos finalizan. Dentro de un mismo *Learning Design* puede haber varios *plays* ejecutándose de manera concurrente. En la Ilustración 3.2 se puede ver el modelo conceptual de IMS LD.

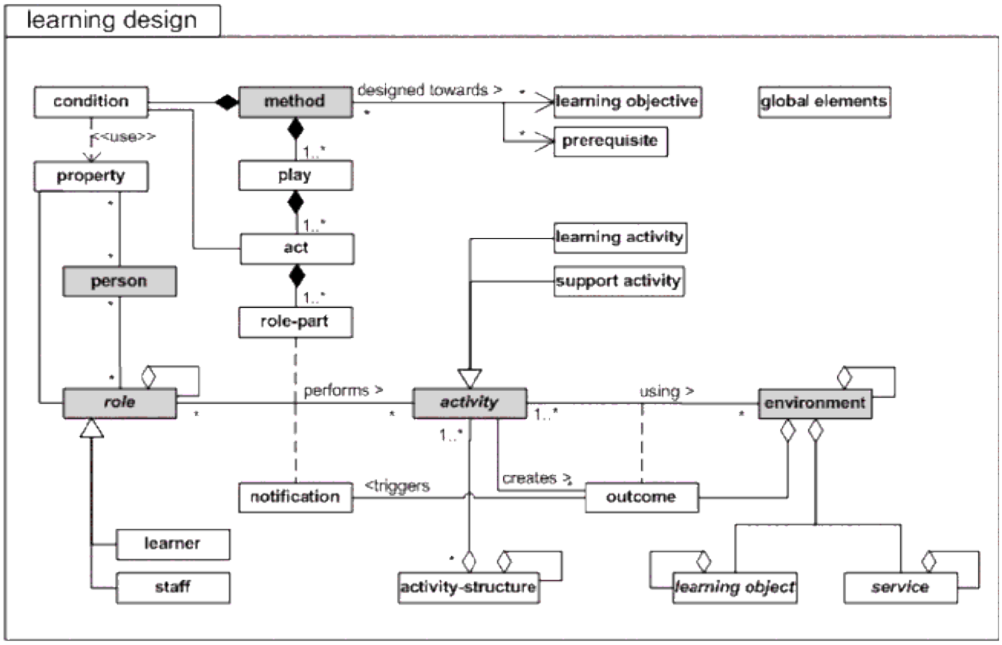


Ilustración 3.2 Modelo Conceptual de Learning Design

A continuación se describen los elementos básicos que conforman una UOL:

- Actores (*person*). Distintas personas o entidades que están involucradas en un proceso de aprendizaje.

- Roles (*role*). Definen las responsabilidades que los distintos actores tendrán en las distintas etapas del proceso de aprendizaje. Un mismo actor puede actuar bajo distintos roles en distintos momentos del proceso de aprendizaje. Por ejemplo, una persona puede ejercer en un momento dado de alumno principiante y más delante de mentor de otros alumnos principiantes.

- Actividades (*activity*). Son procesos educativos atómicos que suceden en un determinado entorno (dentro o fuera del entorno LMS) y que pueden tener asociados uno o varios elementos de contenido que se distribuyen como parte de la UOL.

- Estructuras de actividades (*activity-structure*). Las actividades se pueden agrupar en estructuras, lo que permite referenciar un conjunto de actividades atómicas como una sola entidad. Similarmente, las estructuras de actividades se pueden agrupar en estructuras mayores, dando lugar a otras más complejas formadas por estructuras anidadas.

- Papeles (*role-part*). Son asociaciones entre un rol y una estructura de actividades más o menos compleja. Así, un papel tendría la forma "El actor X realiza la estructura de actividades Y".

- Actos (*act*). Conjunto de papeles que se lanzan simultáneamente (aunque las actividades de los distintos papeles pueden estar secuenciadas internamente de múltiples maneras).

- Obras (*play*). Sucesión de actos que representan la mayor unidad de agrupación en IMS *Learning Design*. Las obras completas se identifican con diseños instructivos completos.

3.3.1.3 NIVELES DE ESPECIFICACIÓN DE IMS LD

La especificación IMS LD plantea un lenguaje potente, aunque es considerado por la comunidad académica como excesivamente complejo de emplear y, sobre todo, de implementar en un LMS.

Para facilitar su adopción progresiva, la especificación define tres niveles distintos de implementación y conformidad, a los que denomina simplemente A, B y C. De este modo, el primer nivel es bastante sencillo de implementar y permite crear diseños instructivos accesibles. Los niveles B y C añaden funcionalidad y potencia, construyendo siempre sobre el nivel anterior.

- Nivel A. Se centra en superar el modelo de único usuario (un alumno trabajando en solitario) reflejado en el resto de las especificaciones IMS. En este primer nivel de especificación se incluyen conceptos básicos como las obras, divididas en actos, en las que los distintos actores interpretan distintos roles.

  La noción de estructuras de actividades, que es la esencia de la definición de los caminos de aprendizaje con ramificaciones, también aparece en este nivel. Con esta información es posible crear UOL en las que se define un proceso colaborativo en el que participan varios actores (tanto alumnos como instructores u otros miembros de apoyo) y se define una secuencia compleja de actividades en las que en algunos casos se le da importancia al orden y en otras no. Lo que no se incluye en este nivel es la posibilidad de modificar y consultar valores, con lo que los flujos de aprendizaje son fijos y el resultado de las distintas actividades no puede afectar al resto.

  Aún así, una implementación que sólo soporte este nivel podría soportar un modelo en el que aparezcan distintos tipos de participantes que realizan distintas actividades en un determinado orden. Por tanto, este nivel ya presenta una aportación sobre el modelo dirigido a un único tipo de usuario y abre la puerta a diseños instructivos basados en los principios del aprendizaje colaborativo.

Por otro lado, dado que otra posible interpretación de los roles es la de distintos perfiles de alumnos, este nivel soportaría modelos educativos en los que distintos tipos de alumno recorren distintos caminos al realizar un determinado curso.

- Nivel B. Las dos aportaciones fundamentales de este nivel son las propiedades y las condiciones. Las primeras son pares atributo-valor que parten de un estado inicial y se modifican a lo largo del proceso de ejecución de la UOL. En cuanto a las segundas, son consultas que se realizan sobre el valor de las propiedades en un momento determinado.

  Este nivel aporta la posibilidad de que el resultado de una actividad genere un cambio en alguna de las propiedades. Por su parte, el resto de actividades pueden estar condicionadas a un cierto valor de las propiedades. En la práctica, esto significa que el resultado de unas actividades puede tener un impacto real en el resto del proceso de aprendizaje, cambiando el camino a seguir o incluso modificando el propio contenido de alguna actividad.

  Adicionalmente, las propiedades pueden ser también externas, es decir, no son modificadas por la propia UOL, sino que son definidas por el propio LMS. Esto significa que en este nivel también se pueden crear UOL que se comporten de manera distinta en función de las exigencias del propio LMS. Un ejemplo común sería que el LMS emplee este mecanismo para quitar del proceso de aprendizaje aquellas actividades inadecuadas para el perfil de los alumnos o que simplemente requieran servicios no implementados por el entorno de aprendizaje (como por ejemplo, un foro de discusión).

  Estos cambios son síncronos, es decir, las actividades se ejecutan en un determinado orden y esperan a que la actividad anterior termine antes de comenzar su ejecución.

- Nivel C. Este nivel introduce un mecanismo de notificación o de envío de mensajes entre las distintas actividades. Esto significa que una actividad puede estar ejecutándose en unas determinadas condiciones y, en un momento no predecible, recibir un mensaje desde otra actividad o desde el propio LMS que afecte a la ejecución de la actividad inicial.

  Esto permite soportar flujos de aprendizaje modificables en tiempo real mediante eventos. Los flujos predefinidos se sustituyen por actividades que se disparan, modifican o interrumpen a medida que cambia el estado de la UOL.

  Dado que en estos procesos de aprendizaje normalmente hay varios individuos, el camino que se seguirá y el orden de ejecución de las actividades ya no es predecible, pues es alterado por la acción de los distintos roles.

Con todo esto, dentro de la misma UOL es posible implementar diferentes formas de interacción y órdenes de presentación de materiales para diferentes perfiles de alumnos. Esto contribuye a incrementar la reusabilidad de las UOL, ya que su diseño se adecuará a un mayor rango de posibles escenarios. En cualquier caso, la implementación de la lógica de adaptación dentro de la propia UOL comprende ciertas limitaciones, ya que una vez terminada la etapa de diseño, el manifiesto no se puede modificar para incluir nuevas estrategias de adaptación. Esto significa que todas las posibles variaciones del proceso se deben especificar de manera previa a la ejecución de la UOL, o esta se deberá rediseñar cada vez que sea necesario actualizar la lógica de la adaptación.

La especificación incluye un modelo de información que detalla en lenguaje natural las características de todos los elementos del *Learning Design*, incluyendo las restricciones que cada uno de ellos debe cumplir y, por otra parte, un esquema XML donde el vocabulario y las relaciones entre los elementos quedan definidos.

3.3.1.4 AUTORÍA DE CONTENIDOS

Dado lo complejo del tema a tratar, la especificación IMS LD resulta muy complicada para un autor de contenidos. Se emplea un lenguaje XML muy verboso y con muchas referencias cruzadas, lo cual hace que resulte poco asequible su uso directo por parte de un autor de contenidos que carezca de una gran experiencia y fluidez en el uso de tecnologías XML.

Actualmente existen varias aplicaciones que permiten la autoría de UOL. La más completa es el editor desarrollado dentro del marco del proyecto RELOAD [13], el cual permite la edición de manifiestos de nivel A, B y C mediante una interfaz gráfica. El diseñador debe rellenar uno por uno los distintos formularios correspondientes a los distintos elementos del *Learning Design*, asociando diferentes recursos a los mismos.

En la Ilustración 3.3 se puede ver una imagen de este editor.

Otra de estas herramientas es el editor de modelado MOT PLUS [14], desarrollado por el centro de investigación LICEF de la Universidad a Distancia de Montreal, que permite la creación de manifiestos de nivel A empleando representaciones gráficas.

Por último, CopperAuthor [15] implementa un editor de IMS LD, soportando también los tres niveles y con el factor adicional de haber sido desarrollado por la propia *Open Univerisity of the Netherlands* (creadores del lenguaje EML original y participantes activos en el grupo de trabajo responsable de IMS LD) en conjunción con la *Open University of the United Kingdom*. Es un programa de código abierto.

En la Ilustración 3.4 se puede ver una imagen de este editor.

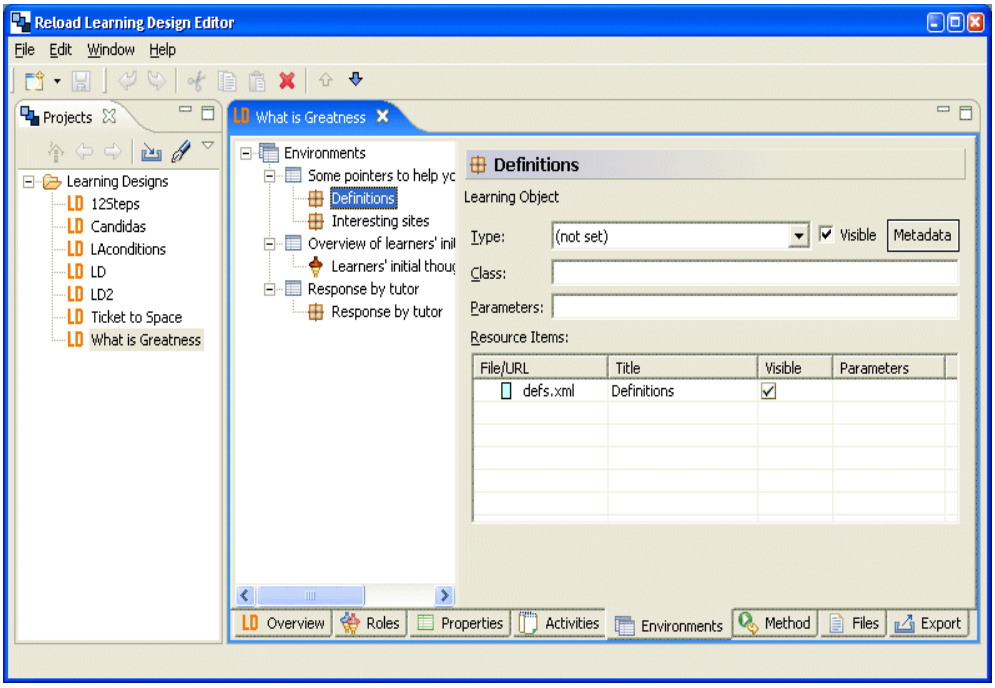


Ilustración 3.3 Editor de IMS LD del proyecto RELOAD

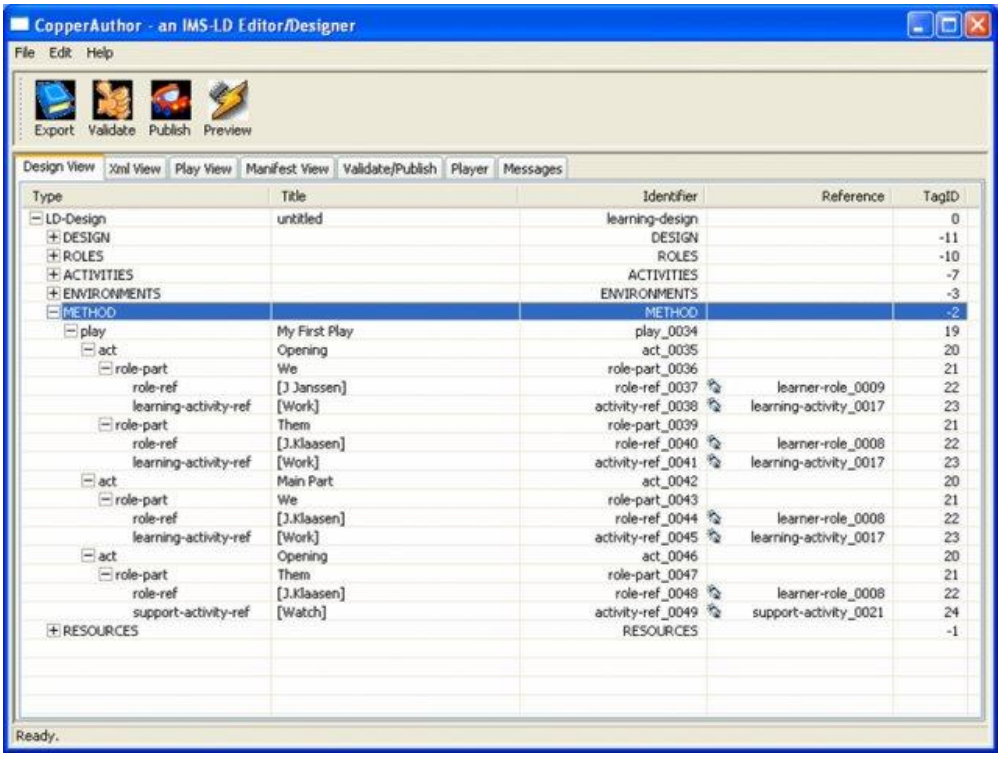


Ilustración 3.4 Editor de IMS LD CopperAuthor

3.3.1.5 REPRODUCTORES

Una vez que la UOL ha sido construida, es necesario disponer de un *software* que se encargue de interpretar su contenido y presentar las diferentes actividades y recursos a los participantes del proceso de aprendizaje, controlando al mismo tiempo sus interacciones.

Dentro de este tipo de *software*, denominado *Learning Design Player*, la OUNL ha desarrollado una aplicación *open source*, llamada CopperCore [15], que es capaz de interpretar los tres niveles del IMS *Learning Design*. Algunas de sus características son:

- Esconde la complejidad de la ejecución para que sea utilizada por los desarrolladores.
- No proporciona ninguna interfaz.
- Basada en tecnología J2EE.
- Utiliza Java y XML como base para la programación e intercambio de información entre entidades.
- Utiliza variables como mecanismo de almacenamiento de datos.
- Proporciona tres API y un conjunto de pruebas.
- Es independiente de la plataforma.
- Ofrece soporte para bases de datos relacionales (MS SQL Server, Postgre SQL y HSQLDB).
- Preparado para usar un servidor de aplicaciones JBoss, si bien puede ejecutarse en otros servidores sin problema.
- *Software* autorizado por GNU GPL.

Esta aplicación ha sido diseñada para ser integrada dentro de un sistema *eLearning* e incluye herramientas que permiten la validación de las UOL, así como una interfaz y línea de comandos a través de la cual es posible crear y borrar instancias de una UOL, así como dar de alta usuarios y asignarles determinados roles.

En la Ilustración 3.5 se puede ver una imagen de este reproductor.

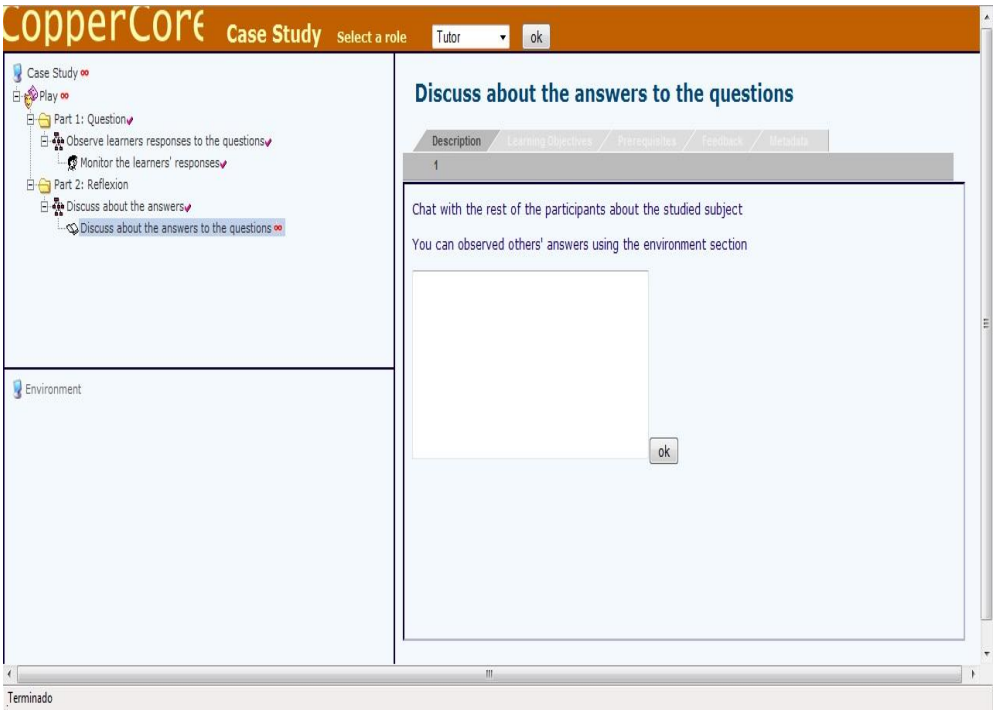


Ilustración 3.5 Reproductor de Learning Design de CopperCore

El proyecto RELOAD [13] también cuenta con un reproductor de *Learning Design*, desarrollado por *The University of Bolton*, basado en el motor de CopperCore. Sus principales características son:

- Ofrece una interfaz de gestión sencilla.
- Lanza y despliega automáticamente CopperCore mediante un servidor JBoss.
- La interfaz permite la importación/borrado de *Learning Design* en el motor CopperCore sin tener que usar la línea de comandos.
- Lee automáticamente un *Learning Design* y rellena CopperCore con una ejecución por defecto y un usuario activo para cada rol encontrado en el manifiesto. También ofrece la opción de crearse uno propio.
- Fácil ejecución. Basta con pinchar en un rol dentro de la interfaz de gestión para que se cargue en el navegador.
- Escrito en Java y disponible para plataformas Windows, Linux y Mac OSX.
- Es gratuito.

En la Ilustración 3.6 se puede ver una imagen de este reproductor.

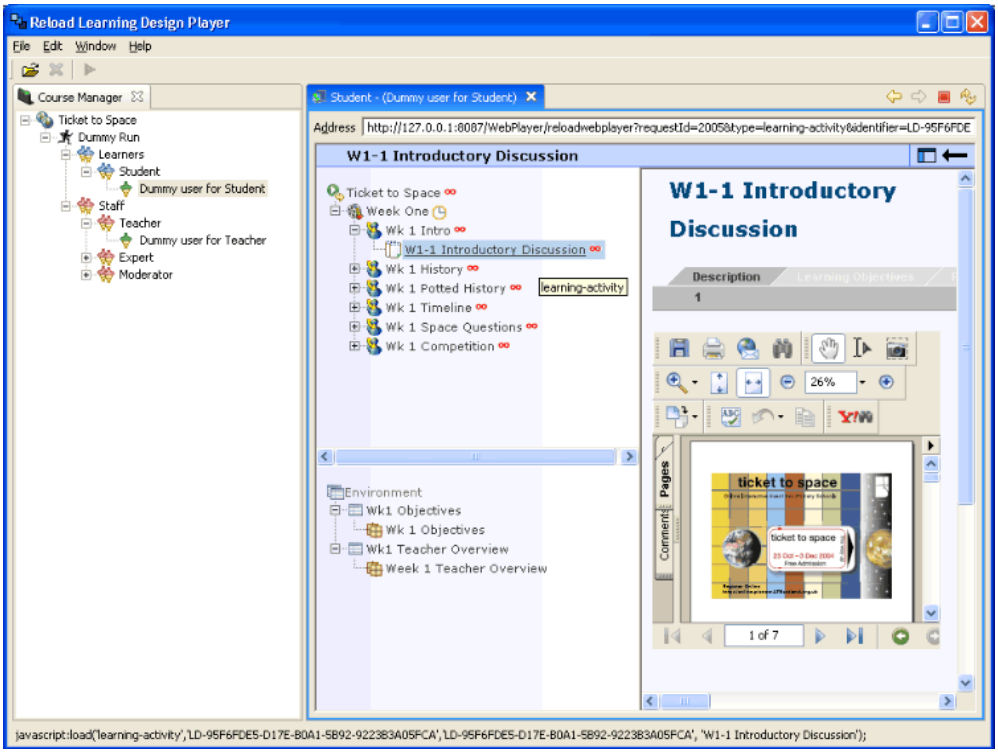


Ilustración 3.6 Reproductor de Learning Design del proyecto RELOAD

Por último, se encuentra el reproductor Sled (*Service Based Learning Design*) [16], que es una versión de CopperCore en la que se mejora la interfaz ofrecida al usuario. Ha sido desarrollado por *The Open University*. Las principales ventajas ofrecidas por este reproductor en detrimento de CopperCore son las siguientes:

- Ofrece una interfaz Web para la gestión de usuarios y ejecuciones en lugar de la herramienta ofrecida por CopperCore consistente en un intérprete de comandos.

- Permite personalizar el diseño y/o la distribución de cada curso configurado en el motor CopperCore. Los nuevos diseños pueden ser creados usando XSL y XML.

- Da acceso a un foro de ejemplos y servicios de búsqueda para demostrar cómo pueden ser implementados.

- Está disponible la capacidad de cambiar los proveedores de servicio usando el concepto *CopperCore Service Integration* (CCSI).

- Dispone de un *plugin* que permite la integración con MOODLE (*Modular Object-Oriented Dynamic Learning Environment*) [17].

En la Ilustración 3.7 se puede ver una imagen de este reproductor.

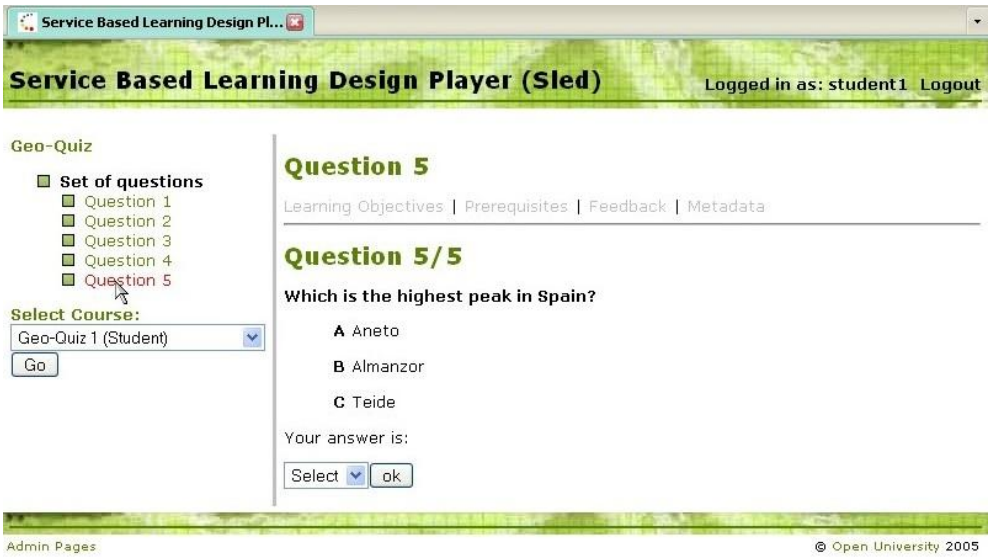


Ilustración 3.7 Reproductor de Learning Design de SLED

3.4 MODELO DE ADAPTACIÓN PARA IMS LD: ADAPTATION POKES

El profesor de la Universidad Carlos III de Madrid Telmo Agustín Zarraonandia Ayo expone en su tesis doctoral "Adaptaciones de unidades de aprendizaje en tiempo de ejecución" [18] un modelo para solucionar los problemas de modificar una UOL en tiempo de ejecución, basado en *Adaptation Pokes*. Un *Adaptation Poke* es una descripción de pequeñas adaptaciones sobre algunos de los elementos de un proceso de aprendizaje. La definición de un *Adaptation Poke* implica:

- Descripción de la acción de la adaptación. Se define el cambio que se va a aplicar a la definición original de la UOL (modificación, borrado o adición). Es la única parte obligatoria en la definición.

- Descripción de nuevos elementos. Cuando la adaptación implica la introducción de nuevos elementos en el diseño del proceso de aprendizaje, será necesario proveer la definición de dichos elementos.

- Nuevos recursos. La adaptación puede requerir la introducción de nuevos contenidos o la sustitución de alguno ya existente.

Una adaptación estará constituida por la aplicación de uno o más *Adaptation Pokes*. Un *Adaptation Poke* está a su vez constituido por una serie de acciones adaptativas o modificaciones que tras ser procesadas en su totalidad dejan la UOL en un estado consistente, es decir, que no infringe ninguna de las restricciones de la especificación empleada para la definición de su manifiesto.

Para llevar a cabo la adaptación de una UOL en ejecución, los *Adaptation Poke* deberán ser publicados indicando la UOL, ejecución y usuarios para los cuales se debe aplicar la modificación.

3.4.1 TIPOS DE ADAPTACIONES

En principio, todos los elementos descritos en el diseño de un proceso de aprendizaje son susceptibles de ser adaptados. Sin embargo, el mecanismo de adaptación propuesto únicamente tiene sentido cuando implica pequeñas variaciones en los diseños. Cuando el objetivo es llevar a cabo modificaciones importantes en la estructura del proceso, como por ejemplo la inclusión de nuevos roles, la estrategia pedagógica implementada puede sufrir serias modificaciones por lo que resulta más conveniente llevar a cabo un rediseño completo del proceso en el que se estudien y analicen los efectos del cambio.

Se pueden distinguir dos tipos de adaptaciones según el objeto de la modificación:

- Adaptaciones estáticas o estructurales. Son las que modifican la definición del proceso de tal forma que añaden o eliminan elementos a su estructura o bien modifican la definición de los ya existentes. Ejemplos de adaptaciones de este tipo serían la introducción de material auxiliar, la supresión de ciertas partes del proceso o la sustitución de un recurso por otro.

- Adaptaciones dinámicas o de estado. Son las que modifican el estado de los elementos del proceso en un momento particular de una ejecución. Como ejemplos de este tipo se considerarían los cambios de estado de una actividad de incompleta a completa o la modificación de la visibilidad de un recurso.

3.4.2 TIPOS DE ADAPTATION POKES

Dependiendo del propósito de la adaptación, podemos clasificar los *Adaptation Pokes* en tres grupos:

- Modificación. Dan un valor a un atributo de un elemento del proceso. Algunos ejemplos pueden ser cambiar el estado de visibilidad o en la completitud de una actividad, o cambiar el recurso al que hace referencia una actividad.

- Adición. Son los que introducen nuevos elementos en la estructura de una UOL, ya sea añadiendo actividades o *environments*. Normalmente requerirá la correspondiente definición del nuevo elemento en un manifiesto, y en ocasiones indicar también el elemento padre y la posición que ocupará dentro de la secuencia de hijos del padre.

- Borrados. Son los que suprimen un elemento de una estructura concreta dentro del proceso, ya sea borrando actividades existentes o *environments*.

En la Ilustración 3.8 se pueden ver los elementos que componen un *Adaptation Poke*.

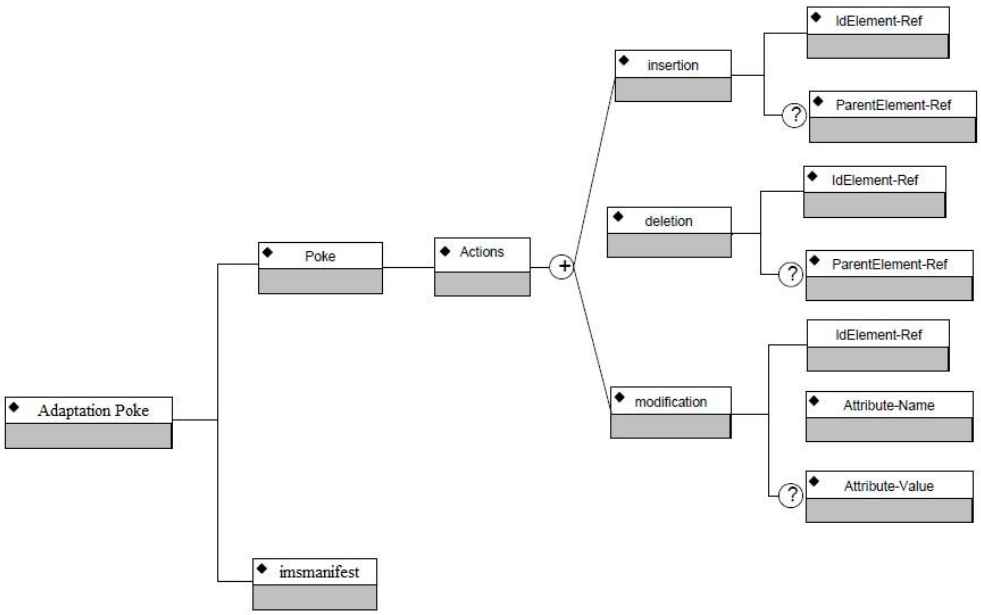


Ilustración 3.8 Diagrama con los componentes de un Adaptation Poke

Un *Adaptation Poke* será un archivo comprimido con la extensión .zip, y estará compuesto por:

- Un archivo poke.xml, donde se incluirán las siguientes referencias:
  - ✓ Identificador de la UOL en la que se hará la adaptación.
  - ✓ Acción a realizar: *insertion*, *modification* o *deletion*.
  - ✓ IdElement-Ref: identificador del elemento a cambiar (en caso de *adition* o *deletion*).
  - ✓ IdElement-Ref: identificador del padre del elemento a cambiar (en caso de *modification*).
  - ✓ ParentElement-Ref: identificador del padre del elemento a cambiar (en caso de *adition* o *deletion*).
  - ✓ Attribute-Name: nombre del atributo que se desea cambiar (en caso de *modification*).
  - ✓ Attribute-Value: valor que se le va a dar al atributo.
- En caso de que el archivo poke.xml sea de tipo *adition*, será necesario un archivo imsmanifest.xml, donde estará definida la nueva UOL, siguiendo las especificaciones de IMS LD.

4 ANÁLISIS

En esta sección se pretende determinar cuáles son las necesidades que deben ser cubiertas y satisfechas por el nuevo sistema. Para esto ha sido necesario establecer una serie de reuniones con el cliente (en este caso el profesor Telmo Agustín Zarraonandia Ayo ejerce este rol) en las que se han identificado las metas globales, las necesidades, los requerimientos y todo aquello que pueda ayudar a la identificación y desarrollo de este proyecto. Este análisis se va a dividir en tres partes: características del entorno, especificación de requisitos y especificación de casos de uso.

4.1 CARACTERÍSTICAS DEL ENTORNO

Antes de comenzar con la especificación de los requisitos del sistema es necesario indicar una serie de características que deben ser tenidas en cuenta para el desarrollo del sistema, y que tienen que ver con los *Adaptation Poke*. No se consideran requisitos como tal porque no pueden ser reflejados en una matriz de trazabilidad entre requisitos y pruebas de evaluación.

- Un *Adaptation Poke* tiene la extensión zip.
- Los archivos contenidos en un *Adaptation Poke* tiene la extensión xml.
- Un *Adaptation Poke* está compuesto al menos de un archivo (poke.xml).
- Los archivos contenidos en un *Adaptation Poke* son archivos de cambios sobre una UOL (poke.xml) o manifiestos (imsmanifest.xml).
- El archivo poke.xml contiene la acción de la adaptación, y puede ser de tipo cambio (*modification*), adición (*adition*) o borrado (*deletion*).
- El archivo imsmanifest.xml sigue la especificación IMS LD.
- Un archivo poke.xml de cambios debe incluir el identificador del elemento y el atributo sobre el que se hace el cambio, además del nuevo valor.
- Un archivo poke.xml de adición debe incluir el identificador del elemento al que va a ser añadido y el nombre del nuevo elemento.
- Un archivo poke.xml de borrado debe incluir el identificador del elemento que va a ser borrado y el del elemento que lo contiene.
- Un *Adaptation Poke* debe contener un archivo imsmanifest.xml cuando el archivo poke.xml sea de adición.

- El lenguaje de programación es Java, la plataforma de programación J2EE y la herramienta de compilación Ant.

4.2 ESPECIFICACIÓN DE REQUISITOS

El principal objetivo de las reuniones mantenidas con el cliente es el de obtener un conjunto de requisitos detallado, no ambiguo y completo que sirva de base para los procesos posteriores y que a su vez sea estable para no tener problemas en otras fases que conlleven una revisión y modificación de los mismos.

Se pueden distinguir dos tipos de requisitos: funcionales o no funcionales.

- Requisito funcional. Expresa una capacidad del sistema, es decir, los servicios que la aplicación debe proporcionar.

- Requisito no funcional. Impone restricciones, tanto en el producto desarrollado como en su proceso de desarrollo. Estos se pueden dividir, a su vez, en:

  - ✓ Operativo. Se refieren a necesidades relacionadas con el entorno, tanto físico como técnico (*hardware*, *software*, normas, ubicación de los equipos, etc.), en el que debe funcionar el sistema.

  - ✓ Seguridad. Están relacionados con la seguridad de la aplicación y el tratamiento de la información que se almacena en la misma.

Cada requisito identificado debe tener los siguientes atributos:

- Identificador. Este campo facilitará la posterior trazabilidad. Estará compuesto de la siguiente manera: R<grupo><tipo>-<número>, donde <grupo> podrá ser F en caso de ser funcional o NF en caso de ser no funcional; <tipo> se rellenará en caso de tratarse de un requisito no funcional, y podrá ser R si es de rendimiento, S si es de seguridad, O si es operativo y N si es normativo; <número> será una cifra de tres dígitos, comenzando por 001.

- Descripción. Se detallará brevemente el fin del requisito.

- Necesidad. Este atributo indica lo importante que es un requisito para el cliente. Puede tomar los valores de esencial, deseable u opcional.

- Prioridad. Indica la importancia del requisito, imprescindible para definir una estrategia de desarrollo. Puede tomar los valores de alta, media o baja.

- Estabilidad. Indica si un requisito es susceptible a sufrir cambios a lo largo del desarrollo del proceso. Puede tomar los valores de alta, media o baja.

- Fuente. Define el origen del requisito. En este caso siempre es el cliente.

- Claridad. Indica si un requisito puede llevar a confusión, es decir, hace referencia a su ambigüedad. Puede tomar los valores sí o no.

- Verificabilidad. Indica si se puede comprobar indiscutiblemente que el requisito se ha incorporado al *software*. Puede tomar los valores sí o no.

Una vez definidos los diferentes tipos de requisitos que pueden existir y los atributos que debe incluir cada uno, se detallan los requisitos obtenidos.

4.2.1 REQUISITOS FUNCIONALES

Estos requisitos harán referencia a las distintas funcionalidades que debe ofrecer el sistema. Se va a hacer una división en apartados para que los requisitos queden estructurados atendiendo al tipo de funcionalidad que van a orecer. Estos tres tipos son: requisitos funcionales de modificación, requisitos funcionales de adición y requisitos funcionales de borrado.

4.2.1.1 REQUISITOS FUNCIONALES DE MODIFICACIÓN

RF-001			
Descripción	Se permitirá cambiar el *environment* de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Media	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.1 Requisito Funcional RF-001

RF-002			
Descripción	Se permitirá cambiar el *environment* de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Media	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.2 Requisito Funcional RF-002

RF-003			
Descripción	Se permitirá cambiar el *environment* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Media	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.3 Requisito Funcional RF-003

RF-004			
Descripción	Se permitirá cambiar el título de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.4 Requisito Funcional RF-004

RF-005			
Descripción	Se permitirá cambiar el tipo de una *activity-structure*.		
Necesidad	Deseable	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.5 Requisito Funcional RF-005

RF-006			
Descripción	Se permitirá cambiar el título de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.6 Requisito Funcional RF-006

RF-007			
Descripción	Se permitirá cambiar el título de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.7 Requisito Funcional RF-007

RF-008			
Descripción	Se permitirá cambiar la referencia al recurso utilizado en una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.8 Requisito Funcional RF-008

RF-009			
Descripción	Se permitirá cambiar la referencia al recurso utilizado en una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.9 Requisito Funcional RF-009

RF-010			
Descripción	Se permitirá cambiar el estado de visibilidad de una *learning-activity*.		
Necesidad	Deseable	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.10 Requisito Funcional RF-010

RF-011			
Descripción	Se permitirá cambiar el estado de visibilidad de una *support-activity*.		
Necesidad	Deseable	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.11 Requisito Funcional RF-011

RF-012			
Descripción	Se permitirá completar una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.12 Requisito Funcional RF-012

RF-013			
Descripción	Se permitirá completar una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.13 Requisito Funcional RF-013

4.2.1.2 REQUISITOS FUNCIONALES DE ADICIÓN

RF-014			
Descripción	Se permitirá añadir un nuevo *learning-object* al *environment* de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.14 Requisito Funcional RF-014

RF-015			
Descripción	Se permitirá añadir un nuevo *service* al *environment* de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.15 Requisito Funcional RF-015

RF-016			
Descripción	Se permitirá añadir un nuevo *learning-object* al *environment* de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.16 Requisito Funcional RF-016

RF-017			
Descripción	Se permitirá añadir un nuevo *service* al *environment* de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.17 Requisito Funcional RF-017

RF-018			
Descripción	Se permitirá añadir un nuevo *learning-object* al *environment* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.18 Requisito Funcional RF-018

RF-019			
Descripción	Se permitirá añadir un nuevo *service* al *environment* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.19 Requisito Funcional RF-019

RF-020			
Descripción	Se permitirá añadir una nueva *learning-activity* a una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.20 Requisito Funcional RF-020

RF-021			
Descripción	Se permitirá añadir una nueva *support-activity* a una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.21 Requisito Funcional RF-021

4.2.1.3 REQUISITOS FUNCIONALES DE BORRADO

RF-022			
Descripción	Se permitirá borrar un *learning-object* del *environment* de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.22 Requisito Funcional RF-022

RF-023			
Descripción	Se permitirá borrar un *service* del *environment* de una *learning-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.23 Requisito Funcional RF-023

RF-024			
Descripción	Se permitirá borrar un *learning-object* del *environment* de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.24 Requisito Funcional RF-024

RF-025			
Descripción	Se permitirá borrar un *service* del *environment* de una *support-activity*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.25 Requisito Funcional RF-025

RF-026			
Descripción	Se permitirá borrar un *learning-object* del *environment* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.26 Requisito Funcional RF-026

RF-027			
Descripción	Se permitirá borrar un *service* del *environment* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.27 Requisito Funcional RF-027

RF-028			
Descripción	Se permitirá borrar una *learning-activity* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.28 Requisito Funcional RF-028

RF-029			
Descripción	Se permitirá borrar una *support-activity* de una *activity-structure*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Baja	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.29 Requisito Funcional RF-029

4.2.1.4 OTROS REQUISITOS FUNCIONALES

RF-030			
Descripción	Se permitirá la publicación de *Adaptation Pokes*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.30 Requisito Funcional RF-030

4.2.2 REQUISITOS NO FUNCIONALES

Los requisitos de esta sección abarcan aquellos que hacen referencia a las restricciones del sistema. Se van a dividir en dos tipos: requisitos no funcionales operativos y requisitos no funcionales de seguridad.

4.2.2.1 REQUISITOS NO FUNCIONALES OPERATIVOS

RNFO-001			
Descripción	Un *Adaptation Poke* actuará sobre una UOL ya publicada.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.31 Requisito No Funcional Operativo RNFO-001

RNFO-002			
Descripción	Después de aplicar un *Adaptation Poke* sobre una UOL, el motor de ejecución se encargará de mostrar los cambios a los participantes en el proceso a través de un navegador web.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.32 Requisito No Funcional Operativo RNFO-002

RNFO-003			
Descripción	Los *Adaptation Poke* deberán ser publicados indicando la UOL, la ejecución y los usuarios para los cuales se debe aplicar la modificación.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Media	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.33 Requisito No Funcional Operativo RNFO-003

RNFO-004			
Descripción	Se usará el intérprete de comandos (herramienta Clicc) para publicar un *Adaptation Poke*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.34 Requisito No Funcional Operativo RNFO-004

RNFO-005			
Descripción	Es necesario que el motor de ejecución esté lanzado para poder publicar un *Adaptation Poke*.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.35 Requisito No Funcional Operativo RNFO-005

4.2.2.2 REQUISITOS NO FUNCIONALES DE SEGURIDAD

RNFS-001			
Descripción	Después de aplicar un *Adaptation Poke*, la UOL quedará en estado consistente, siendo posible continuar con el proceso educativo.		
Necesidad	Esencial	Fuente	Cliente
Prioridad	Alta	Claridad	Sí
Estabilidad	Alta	Verificabilidad	Sí

Tabla 4.36 Requisito No Funcional de Seguridad RNFS-001

4.3 ESPECIFICACIÓN DE CASOS DE USO

Una vez especificados los requisitos extraídos de las reuniones con el cliente es necesario profundizar algo más en el análisis, de manera que se pueda definir la funcionalidad y el comportamiento de la aplicación. Para ello se van a detallar los casos de uso del sistema.

Los requisitos extraídos pueden ser expresados como una interacción entre un agente externo, como por ejemplo el usuario final, y la aplicación, es decir, como un caso de uso. Un caso de uso consiste en la típica interacción entre un actor o tipo de usuario de la aplicación y la aplicación en sí misma. Este actor no tiene por qué ser una persona que interactúa con el sistema, sino que puede tratarse de otro sistema externo.

Para detallar los casos de uso es necesario especificar los siguientes elementos:

- Identificador. Este campo facilitará la posterior trazabilidad. Estará compuesto de la siguiente manera: CU-<número>, donde <número> será una cifra de tres dígitos, comenzando por 001.

- Objetivo. Se detallará brevemente el fin del caso de uso.

- Escenario. Indica cómo un actor interactúa con el sistema y cuál es la respuesta que el sistema le ofrece.

- Precondiciones. Definen las condiciones que se deben cumplir para poder realizar una operación.

- Post-condiciones. Definen en qué estado queda el sistema tras realizar una operación.

Una vez definidos los casos de uso y los atributos que debe incluir cada uno, se detallan los casos de uso realizados.

CU-001		
Objetivo	Publicar un *Adaptation Poke*.	
Escenario		
Acción del actor		Respuesta del sistema
Introducir el comando *uploadpoke <id de la uol> <nombre del Adaptation Poke>* en la herramienta Clicc.		Leer el contenido del *Adaptation Poke* y aplicar las acciones pertinentes en la UOL.
Precondiciones	*Adaptation Poke* bien formado.	
	Motor de ejecución lanzado.	
	La UOL sobre la que se aplican los cambios de estar publicada.	
	La UOL sobre la que se aplican los cambios debe tener usuarios.	
	La UOL sobre la que se aplican los cambios debe tener una ejecución.	
Post-condiciones	Se ven los cambios producidos en el navegador web.	

Tabla 4.37 Caso de uso CU-001

Debido a que el principal objetivo de este Proyecto Fin de Carrera es el de implementar un mecanismo que permita realizar adaptaciones de una UOL en tiempo de ejecución, el único caso de uso existente es el de publicar un *Adaptation Poke*, reflejado en la Tabla 4.39. A simple vista puede parecer muy simple, pero alberga las distintas funcionalidades de que es capaz el mecanismo de adaptación, definidas en los requisitos funcionales de la sección anterior.

Una vez especificado este caso de uso, es conveniente recogerlo en un diagrama de casos de uso mediante UML 2.0, que ilustra de manera gráfica la relación de los usuarios que interactúan con él. El único usuario que se encarga de realizar todos los cambios en las UOL es el profesor. Se puede ver en la Ilustración 4.1.

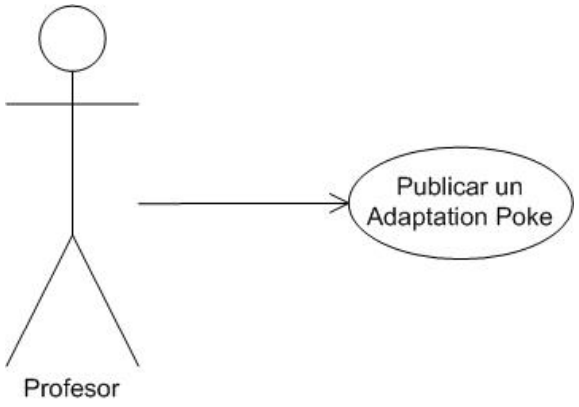


Ilustración 4.1 Diagrama de casos de uso

5 DISEÑO

Una vez concluida la fase de análisis, es necesario sentar las bases que van a permitir construir el sistema. Esto se realiza en la fase de diseño, que consiste en un proceso iterativo en el cual los requisitos extraídos en la fase de análisis se transforman en una representación del *software* con una serie de detalles que permiten su realización física.

Esta fase de va a dividir en dos tareas: la arquitectura del sistema y el diseño a bajo nivel.

5.1 ARQUITECTURA

El diseño de la arquitectura de un sistema consiste en un conjunto de abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del *software* de un sistema de información, cubriendo las necesidades especificadas por el cliente y recogidas por el analista mediante los requisitos. La elección de una arquitectura es un factor determinante a la hora de escoger la tecnología a usar por el sistema, ya que no todas las tecnologías son aptas para todas las arquitecturas, aunque en este caso esta elección no es una opción, pues las tecnologías ya vienen determinadas por CopperCore.

Generalmente no es necesario diseñar una arquitectura nueva por cada sistema a desarrollar, sino que lo normal es escoger una ya conocida, evaluarla y adaptarla a las características del sistema. En esta ocasión, ya que el objetivo del proyecto es el de desarrollar un módulo que permita hacer modificaciones en UOL's que se encuentren publicadas en el motor de ejecución de CopperCore, lo que se busca es tratar de integrar este nuevo módulo con el sistema que ya existe, por lo que no se va a coger una arquitectura ya conocida, como podrían ser una cliente-servidor, una monolítica, una de tres capas, etc.

En este apartado se va a tratar de dar una visión desde el más alto nivel, para ir descendiendo hasta detalles más técnicos que permitan la construcción del nuevo sistema.

Con esta primera descomposición lo que se pretende es dar cierta modularidad al sistema, diferenciando los diferentes componentes e indicando su funcionalidad. Se va a partir de tres componentes principales:

- CopperCore. Es el motor de ejecución sobre el que se va a construir el módulo con las características requeridas. Incluye un reproductor con el que se pueden ver los cursos especificados con IMS LD. Este reproductor incluye la funcionalidad de publicar UOL's, que previamente son validadas. Además, incluye una herramienta en forma de intérprete de comandos, llamada Clicc, que facilita toda la gestión de usuarios y creación de ejecuciones, así como otras cuantas opciones.

- Módulo de adaptación. Se trata del componente que va a encapsular la funcionalidad exigida por el cliente.

- Editor de cursos. Herramienta que sirve para la creación o modificación de cursos basados en IMS LD y que genera los *Adaptation Pokes*. Permite hacer diversas acciones sobre el manifiesto de una UOL, como por ejemplo crear uno nuevo, hacer modificaciones, añadir nuevos elementos, eliminar elementos, etc. Tras realizar todos estos cambios en el manifiesto, se genera un log que servirá como *Adaptation Poke*, el cual permitirá la modificación de una UOL en tiempo de ejecución. La aplicación también crea un archivo con extensión .zip en el que se encuentra el manifiesto editado.

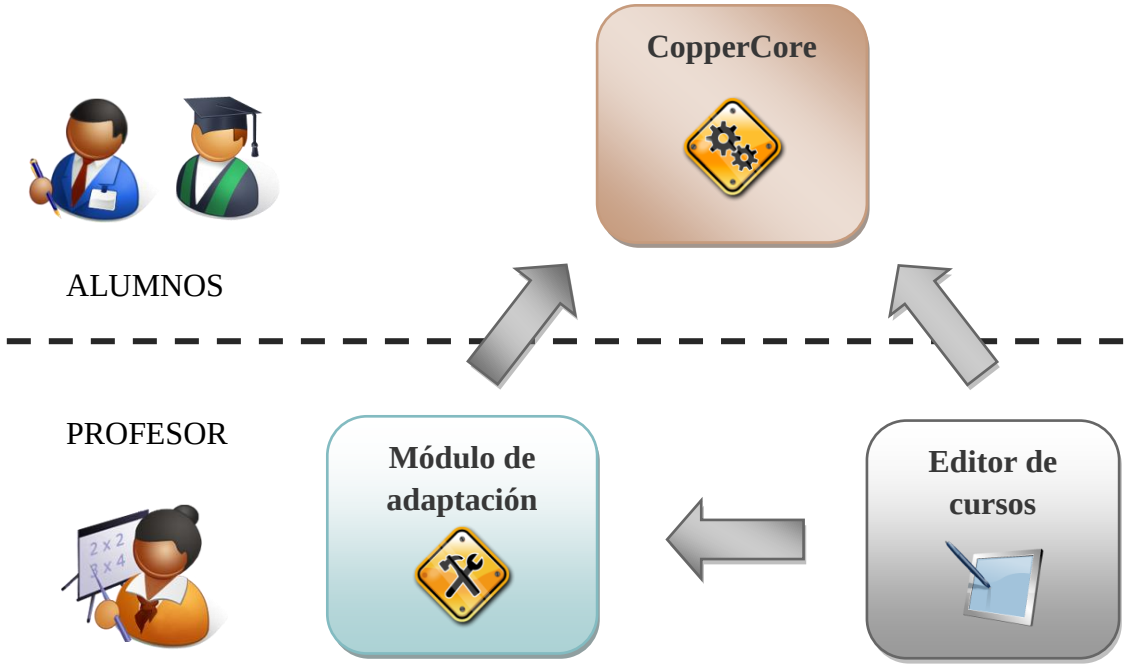


Ilustración 5.1 Arquitectura del sistema

Como se puede observar en la Ilustración 5.1, las dependencias de cada componente de la arquitectura (indicadas mediante flechas) no se cruzan entre sí. El componente editor de cursos se encargará de la creación de *Adaptation Pokes* válidos que serán pasados al módulo de adaptación, el cual, tras procesarlos, se encargará de aplicar los cambios y realizar las acciones pertinentes en el componente CopperCore. El componente editor de cursos también se puede encargar de la generación o modificación de cursos que serán validados por CopperCore antes de su publicación. Una vez publicado el curso, el módulo de adaptación, tras subir un *Adaptation Poke*, se encargará de aplicar los cambios y realizar las acciones pertinentes en el componente CopperCore. Los alumnos sólo ven el componente CopperCore, mientras que el profesor manipula los otros dos.

5.2 DISEÑO DETALLADO

En este apartado se va a profundizar aún más en el diseño del sistema, describiendo detalladamente el comportamiento que deberá tener el componente módulo de adaptación de la arquitectura especificada anteriormente para facilitar su posterior implementación. Para ello vamos a hacer uso de una serie de diagramas que nos ofrezcan una visión gráfica, como pueden ser los diagramas de secuencia y los diagramas de clase. También se dará una visión general de la composición del componente CopperCore, pues el nuevo módulo debe integrarse en él.

5.2.1 RELACIÓN ENTRE LOS COMPONENTES DE LA ARQUITECTURA

Para ilustrar la relación entre los componentes de la arquitectura se va a usar un diagrama de secuencia. Un diagrama de secuencia sirve para modelar la interacción entre los objetos de un sistema, ya que muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Estos diagramas contienen detalles de implementación de los escenarios definidos en los casos de uso, incluyendo los objetos, las clases y los mensajes intercambiados entre ellos. Se va a partir de los casos de uso especificados en el capítulo de análisis y se va a ilustrar un ejemplo que sirva de referencia para ver los objetos implicados en la publicación de un *Adaptation Poke* y la relación que tienen los distintos componentes entre sí.

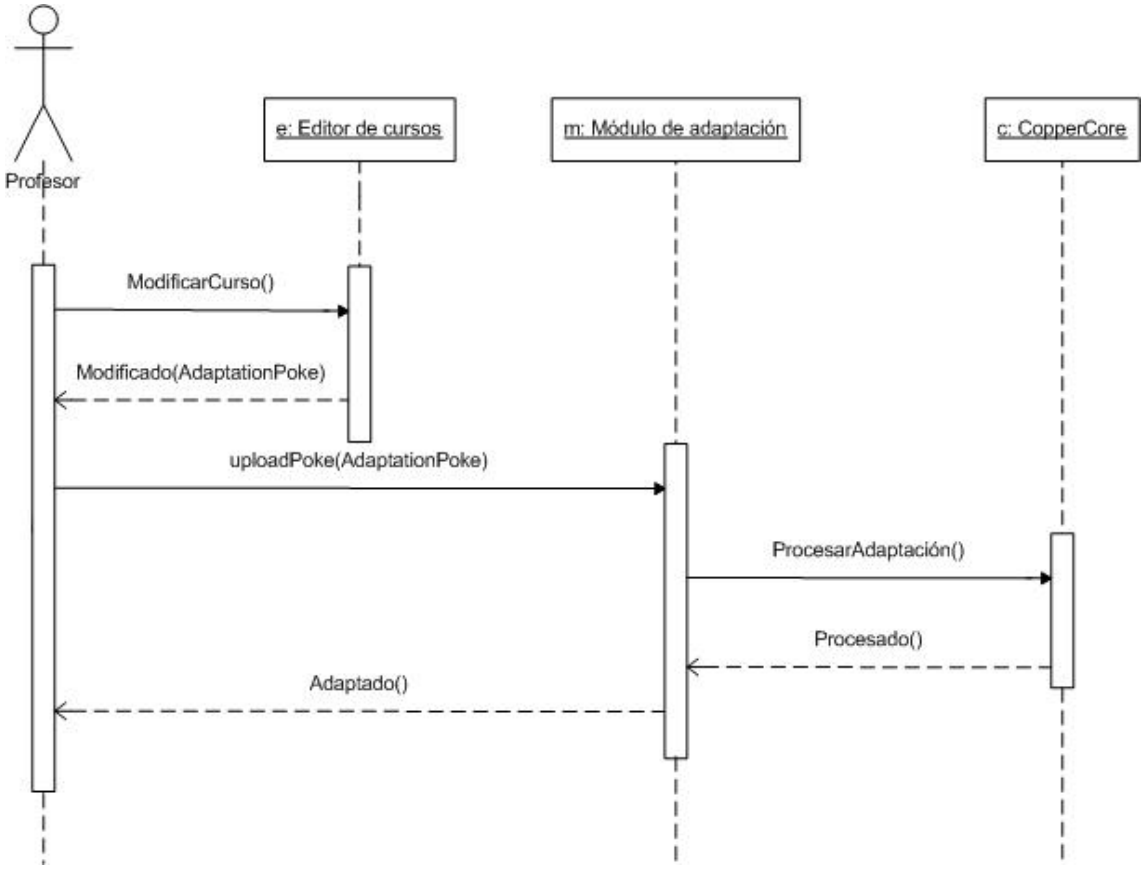


Ilustración 5.2 Diagrama de Secuencia para la publicación de un Adaptation Poke

Como se puede ver en la Ilustración 5.2, el proceso de publicación de un *Adaptation Poke* consiste en modificar un curso, previamente publicado en el motor de ejecución CopperCore, tras lo cual el editor de cursos generará un log que servirá como *Adaptation Poke*. Acto seguido, el profesor lo subirá a través del módulo de adaptación, donde se producirán las comunicaciones con CopperCore y se realizarán los cambios oportunos. Por último, se mostrarán los resultados a todos los usuarios, tanto alumnos como profesores.

5.2.2 COMPONENTE COPPERCORE

CopperCore es el primer motor de ejecución de IMS LD de código abierto que soporta los tres niveles de IMS LD (A, B y C). IMS LD es una especificación compleja y semánticamente rica, por lo que no es fácil darle soporte. IMS LD especifica una plantilla de flujos de trabajo sincronizados y personalizados a través de un curso. En el entorno de ejecución se debe usar esta plantilla para proporcionar al usuario una vista actualizada de su proceso de aprendizaje. Por ejemplo, cuando en el manifiesto del diseño de aprendizaje se especifica un grupo de trabajo donde los alumnos necesitan completar una actividad antes de continuar con la siguiente, el entorno de ejecución debería comprobar esta restricción y debería sincronizar el acceso a la segunda actividad comprobando continuamente si los usuarios ya completaron su primera actividad. Todas estas comprobaciones, sincronizaciones y personalizaciones se conocen como la lógica de negocio de *Learning Design*, y es lo que CopperCore maneja para el desarrollador, consiguiendo hacer transparente estas complejidades.

Hay que mencionar que es necesario conocer el funcionamiento interno de este componente antes de entrar en detalle en el módulo desarrollado, pues sino su comprensión sería muy complicada debido a que se tiene que integrar en CopperCore.

CopperCore está desarrollado en J2EE y proporciona tres API's distintas, *CopperCore API Javadoc* [22], *CopperCore SOAP API Javadoc* [23] y *XML's schemas* [24], y un conjunto de test. Como se puede ver en la Ilustración 5.3, CopperCore está compuesto por una gran cantidad de paquetes que implementan distintas funcionalidades, entre los que cabe destacar:

- Business. Implementa la lógica de negocio de la aplicación.
- Clients. Incluye la herramienta con la que el usuario se encarga de las diferentes gestiones (publicar una UOL, crear usuarios, crear roles, crear ejecuciones…).
- Component. Incluye la lógica de negocio y los diferentes componentes de un *Learning Design*.
- Parser. Se encarga de la lectura de archivos XML.
- Publication. Incluye la herramienta para publicar UOL.

- Validator. Se encarga de la validación de UOL's, comprobando tanto la disponibilidad de los recursos referenciados en el manifiesto como la ausencia de inconsistencias en la definición de este.

- Webplayer. Se encarga de la representación de los datos en un navegador web.

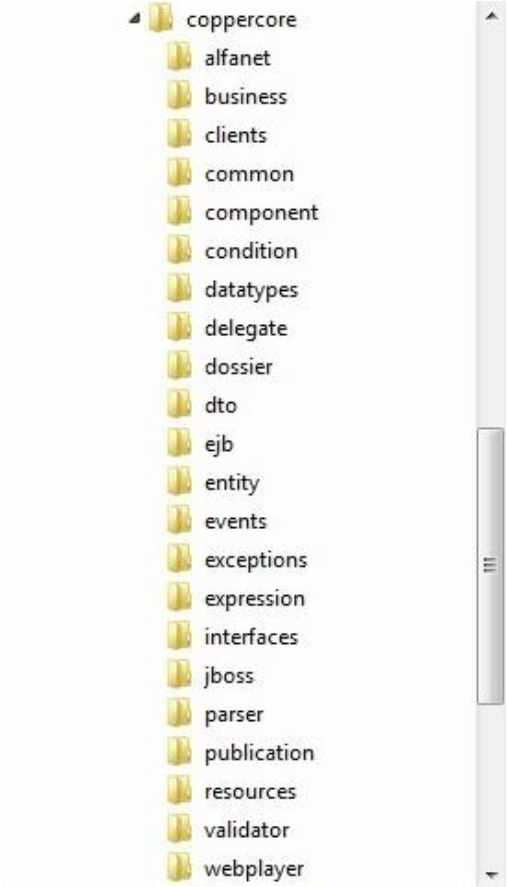


Ilustración 5.3 Estructura de CopperCore

CopperCore no está diseñado para ser usado como un entorno de aprendizaje autónomo, sino que lo está para ser usado como un servicio orientado a *framework*. Este *framework* consiste en diferentes servicios que son combinados para crear un sistema *e-learning* completo. En este contexto, CopperCore se limitaría a implementar un servicio para el procesamiento de de UOL's.

Los servicios ofrecidos por CopperCore están implementados como componentes *software* que proporcionan funcionalidades bien definidas. Para mostrar la funcionalidad de un servicio cada componente proporciona una o más interfaces de programación de aplicaciones bien definidas. Estas interfaces conforman el contrato entre los proveedores y los consumidores de servicio. Para prevenir dependencias no

deseadas entre los servicios, los componentes ocultan su funcionamiento interno a los otros componentes del sistema.

CopperCore implementa cuatro interfaces: *CopperCore*, *CopperCoreAdmin*, *LDEngineDelegate* y *LDCourseManagerDelegate*. Estas interfaces están agrupadas en dos paquetes, cada uno de los cuales con un uso específico. El primer paquete es el de *CopperCoreDelegate*, que contiene las interfaces *LDEngineDelegate* y *LDCourseManagerDelegate*, que a su vez son interfaces de cliente *facade* nativas de Java. Este paquete lo usan los componentes que necesitan llamar a CopperCore a través de una interfaz nativa de Java. Se trata de un paquete cliente *facade* porque el paquete en sí mismo se usa en el contexto de servicios Java que consumen otros servicios CopperCore. El segundo paquete es el paquete *CopperCoreSoap*. Está formado por las interfaces *CopperCore* y *CopperCoreAdmin*. Como el nombre indica, estas interfaces son SOAP y permiten a otros servicios llamar al servicio CopperCore a través de un protocolo SOAP.

Internamente, las interfaces *CopperCoreSoap* hacen uso de las dos interfaces *CopperCoreDelegate* para proporcionar la funcionalidad de CopperCore a los servicios consumidores. Ambos paquetes proporcionan la misma funcionalidad, pero permiten diferentes tipos de servicios consumidores para conectarse a CopperCore.

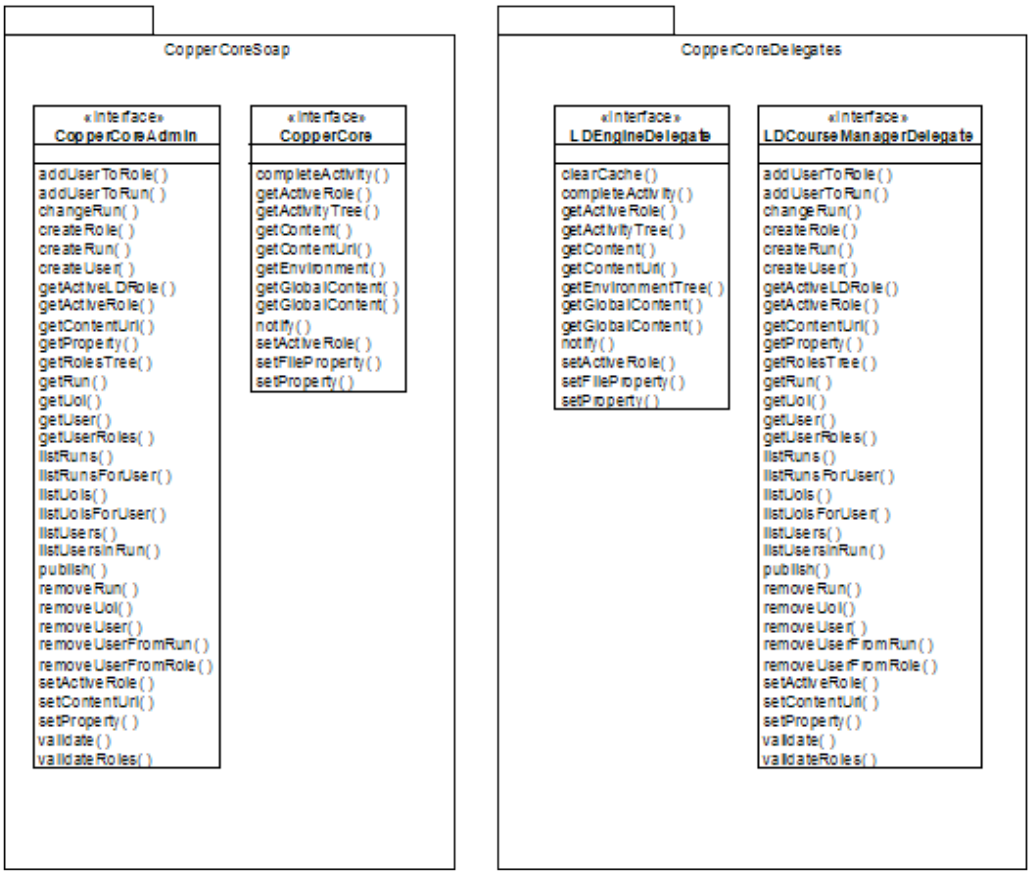


Ilustración 5.4 Interfaces de CopperCore

En la Ilustración 5.4 se pueden ver los diferentes paquetes con sus respectivas interfaces. Para cada interfaz se muestran todas las operaciones que posee. Se puede ver claramente que los dos paquetes de interfaces tienen la misma funcionalidad. Aunque las interfaces contienen las mismas operaciones, sus parámetros y valores de retorno no son siempre los mismos. Esto se debe a las diferencias entre los protocolos de llamada Java y los SOAP. Todas estas interfaces están documentadas usando la herramienta javadoc.

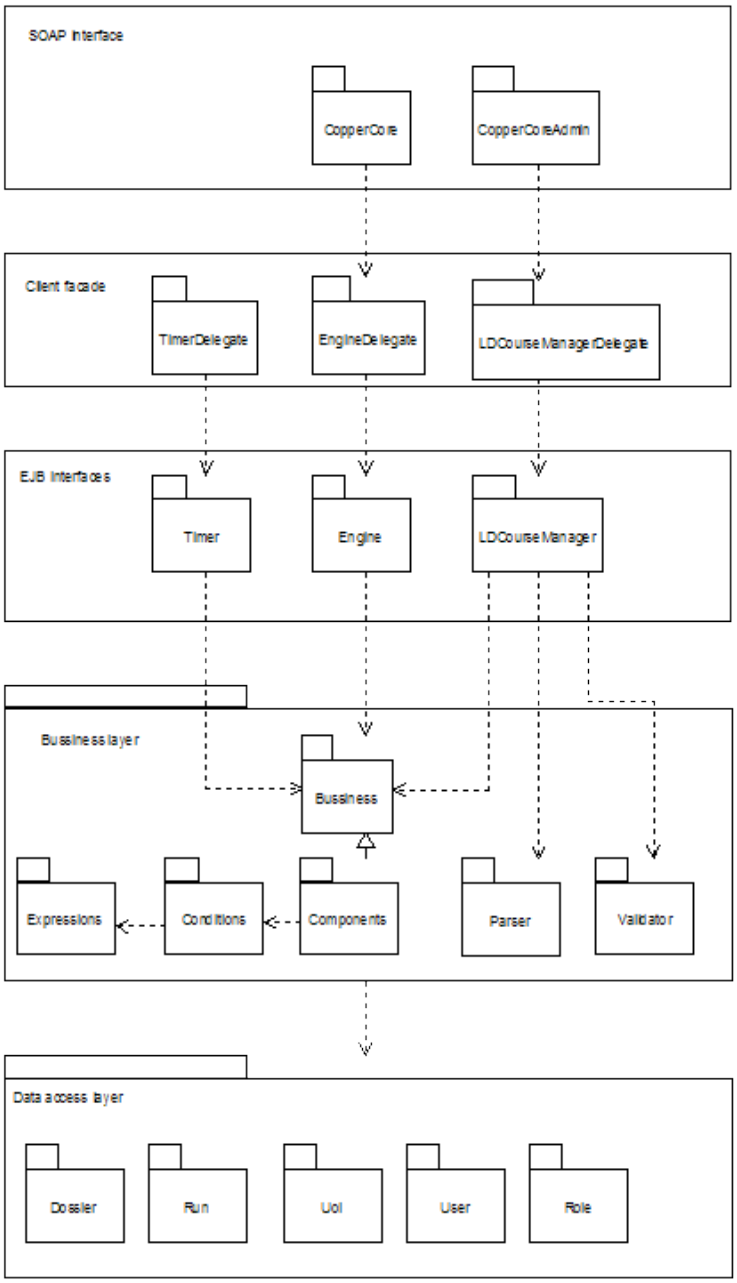


Ilustración 5.5 Modelo UML con la arquitectura de implementación de servicios

La Ilustración 5.5 ofrece una visión de la arquitectura en capas de CopperCore. Las tres capas inferiores forman la implementación real del componente CopperCore, mientras que las otras dos representan los dos paquetes de interfaces que presentan la funcionalidad del servicio *Learning Design*, bien a través de la interfaz nativa de Java (el paquete cliente *facade*), bien a través del protocolo SOAP (el paquete *CopperCoreSoap*).

La capa *Data Access* es la responsable de todas las interacciones directas con la base de datos, y usa EJB de tipo BMP para acceder a la misma. Un componente muy importante de esta capa es el paquete *Dossier*, el cual proporciona acceso de bajo nivel a las propiedades. CopperCore no sólo implementa propiedades de *Learning Design*, sino que también reutiliza este concepto para almacenar el *Learning Design* y para guardar el estado de cada progreso individual a través del *Learning Design*. Los demás componentes tratan con la gestión del curso, la cual implica conceptos como usuarios, ejecuciones, UOL, roles, etc.

La siguiente capa de la arquitectura es la capa *Business*, que contiene todos los componentes que representan la lógica de negocio de CopperCore. Los paquetes *Parser* y *Validator* implementan esta lógica de negocio para importar y para validar un paquete IMS LD. Esta funcionalidad se presenta a través de *CourseManager*. Los otros paquetes de esta capa o bien albergan componentes de gestión de cursos como usuarios, ejecuciones, UOL, etc. o bien representan partes del *Learning Design* que son directa o indirectamente accesibles a través del motor de ejecución. Estos últimos componentes están agrupados en el paquete *Components*. Cada componente contiene toda la lógica de negocio que necesita para adaptarse por sí mismo al perfil del usuario que está accediendo al componente de *Learning Design*. Con este propósito este contenedor hace un uso intensivo del mecanismo de propiedad que contiene su propia lógica de negocio sobre recuperación y almacenamiento de propiedades.

La siguiente capa se llama EJB *Interfaces* y está compuesta por tres EJB de sesión. El primer EJB es el *LDCourseManager*, que se encarga de todas las llamadas administrativas necesarias en la preparación del reparto de una instancia IMS LD. Las interfaces típicas que se ofrecen están relacionadas con la publicación de una instancia XML de *Learning Design*, con la creación de usuarios, con la creación de ejecuciones y con la asignación de roles. Esta capa sólo importa un manifiesto de IMS LD en vez de un paquete de contenido completo. Esto se debe a las restricciones en el servidor J2EE referidas al acceso al sistema de ficheros.

El segundo EJB es el *Engine*. Es el corazón del mecanismo de reparto. Este EJB trata con la personalización de una instancia IMS LD para un determinado usuario en un momento concreto en el tiempo. Las típicas llamadas con las que trabaja están relacionadas con la recuperación de *activityTree* personalizados, *environmentTree* y contenido.

El tercer y último EJB es el *Timer*, que se relaciona con todas las restricciones de tiempo formuladas en el *Learning Design*. Este EJB debería llamarse en intervalos regulares.

La cuarta capa es la *Client facade*. Está formada por delegados de la capa *Business* que implementan una interfaz nativa de Java para los componentes EJB de CopperCore. Estos contienen todo el código para hacer la conexión entre las interfaces EJB, de forma que facilitan el trabajo a los implementadores. Esta capa añade una funcionalidad adicional para validar el contenido de un paquete IMS LD. Se hacen varias comprobaciones para ver si los paquetes están completos, si el *Learning Design* está bien formado y es válido comparado con el esquema y si el *Learning Design* es semánticamente correcto. Esta capa mejora la funcionalidad de importar de la capa anterior, añadiendo una validación explícita antes de la importación del paquete. Esto evita la subida de paquetes con contenido inválido a CopperCore. La auto-importación se mejora permitiendo la importación de paquetes de contenido completos. El *Client facade* es el responsable de desempaquetarlos y de almacenar todos los recursos en una localización específica en el sistema de ficheros. El archivo *imsmanifest*, que es una parte del paquete de contenido, se pasa a la capa EJB, donde es importado a CopperCore.

La quinta y última capa añade una interfaz SOAP a CopperCore presentando las interfaces de los delegados a través de la herramienta SOAP Axis para Java.

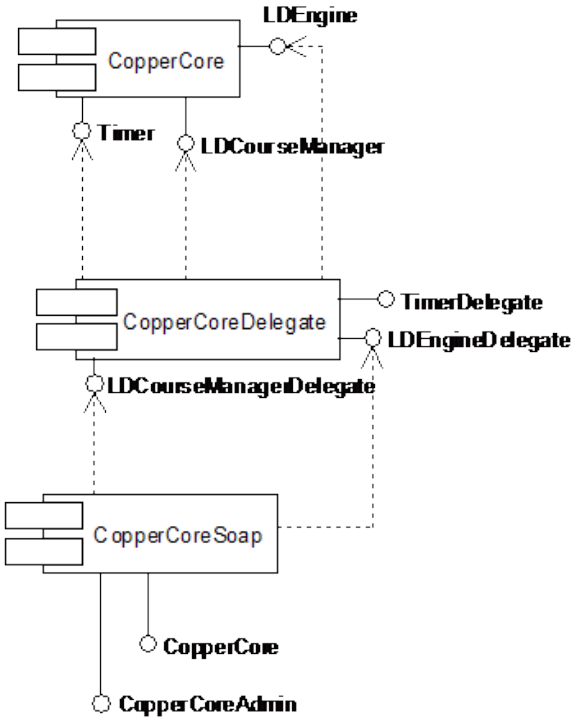


Ilustración 5.6 Diagrama UML de componentes de las interfaces de CopperCore

La Ilustración 5.6 muestra las diferentes interfaces y las relaciones existentes entre ellas. La capa CopperCore proporciona tres interfaces a través de EJB, que son *LDEngine*, *LDCourseManager* y *Timer*. Estas tres interfaces son usadas por *CopperCoreDelegate* encapsulando las complejidades de las llamadas remotas de los EJB. El *CopperCoreDelegate* presenta sus funcionalidades a través de tres API de Java. Se espera que las aplicaciones del cliente nunca llamen a las EJB directamente, sino siempre usando las interfaces *CopperCoreDelegate* o *CopperCoreSoap*. Esta última sólo tiene dos interfaces porque se espera que el temporizador esté disponible en la aplicación servidor de J2EE.

5.2.3 MÓDULO DE ADAPTACIÓN

Una vez estudiada la arquitectura de CopperCore en detalle, ya se puede entrar a especificar el diseño del módulo de adaptación a desarrollar. Para ello se van a señalar los componentes de CopperCore donde se van a integrar las nuevas funcionalidades.

En primer lugar, se tiene que añadir un nuevo servicio a las interfaces existentes de forma que permita publicar un *Adaptation Poke*. El lugar más apropiado para incluirlo se encuentra en aquel o aquellos componentes que implementen las interfaces de CopperCore. Este lugar son las capas *Soap Interface* y *Client facade*, que es donde se encuentran las interfaces nativas de Java (*CopperCore* y *CopperCoreAdmin*) y las que usan el protocolo SOAP (*LDEngineDelegate* y *LDCourseManagerDelegate*). La integración se llevará a cabo dentro del paquete de código fuente de CopperCore *clients*, que a su vez tiene una clase llamada Clicc.java donde se implementan las diferentes interfaces que ofrece CopperCore. La nueva interfaz recibirá el nombre de uploadPoke(). Esto se puede ver en la Ilustración 5.7.

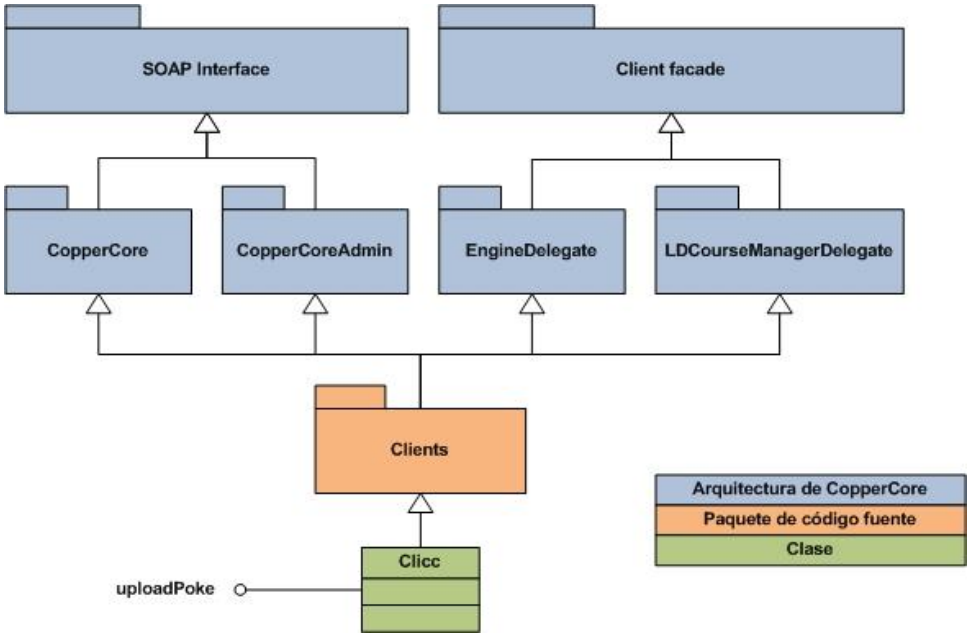


Ilustración 5.7 Diagrama de clases del componente Clients

El siguiente paso en el diseño consiste en añadir todas las nuevas funcionalidades exigidas por el cliente. Tal y como se puede ver en la arquitectura de CopperCore (Ilustración 5.5), toda la lógica de negocio se encuentra albergada dentro de la capa *Business*. Dentro de esta capa, el componente donde residen los diferentes elementos de *Learning Design* se llama *Components*, si bien también son muy necesarios los componentes *Validator* (se comprueba que la UOL es válida) y *Parser* (se encarga de la lectura de los ficheros XML). CopperCore implementa toda esta lógica de negocio a través de los paquetes de código fuente *Component*, *Validator* y *Parser*, por lo que tiene sentido incluir las nuevas clases en alguno de estos paquetes. En la Ilustración 5.8 se pueden ver todas las clases añadidas.

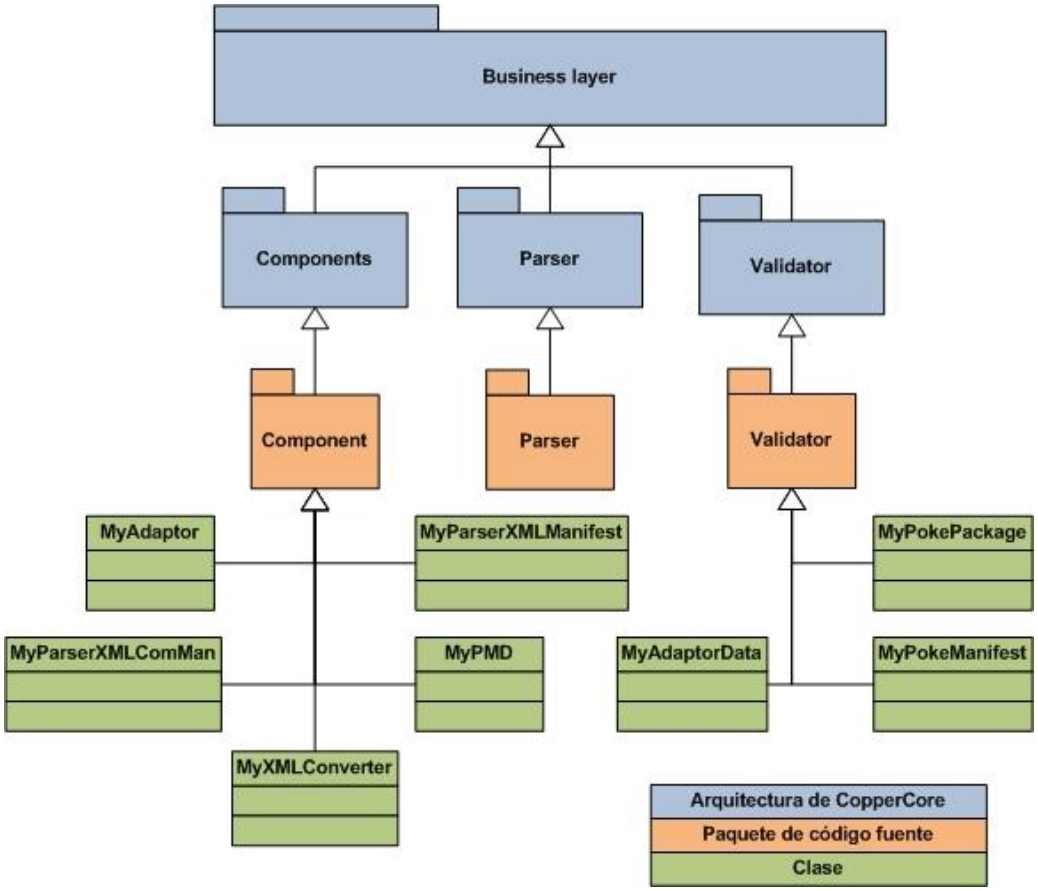


Ilustración 5.8 Diagrama de clases de los componentes Component y Validator

Como se puede apreciar en la Ilustración 5.7 y en la Ilustración 5.8, los tres principales paquetes que se modifican en el código fuente son *Clients*, *Component* y *Validator*. El siguiente paso a establecer en el diseño es la comunicación que se va a producir entre ellos.

Tal y como se puede observar en la Ilustración 5.5, entre las capas *SOAP Interface* y *Client Facade*, que implementan todas las interfaces, y la capa *Bussiness*, que implementa la lógica de negocio, se encuentra la capa *EJB Interfaces*. Por tanto, para

comunicar entre clases del paquete *Clients* y clases de los paquetes *Component* y *Validator* hay que pasar por esta última capa, a través del paquete de código fuente *ejb*.

En la Ilustración 5.9 se puede ver la integración entre la arquitectura de CopperCore y el código fuente.

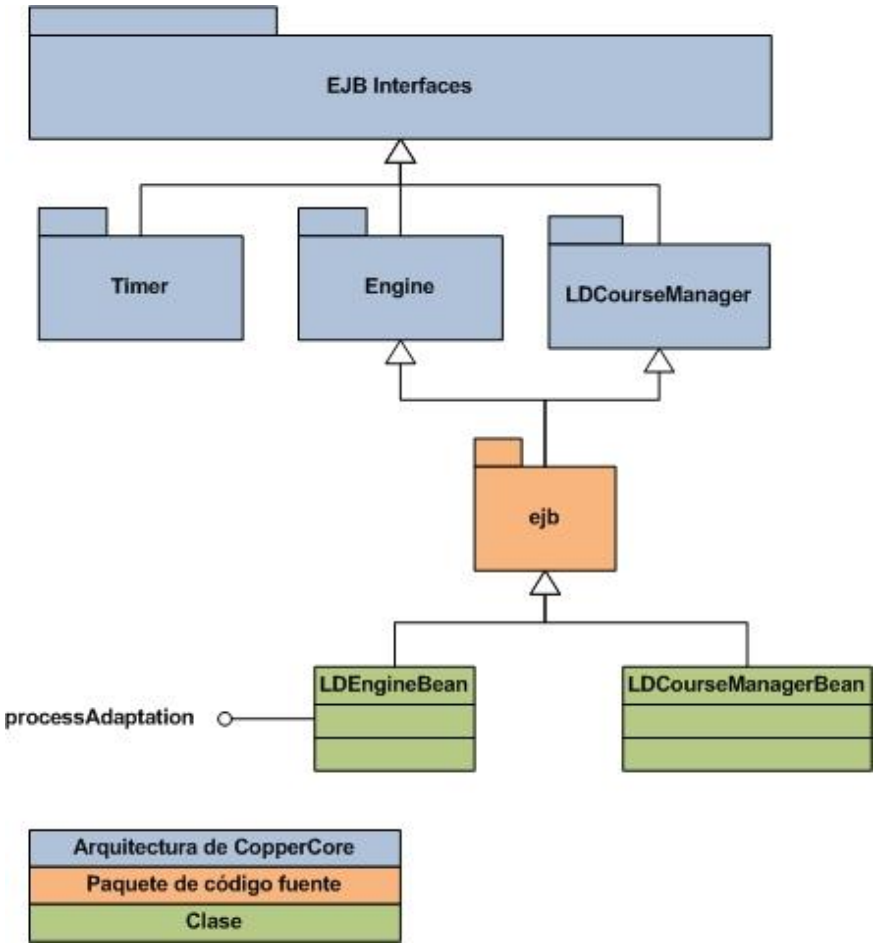


Ilustración 5.9 Diagrama de clases del componente ejb

Por tanto, de la Ilustración 5.7, la Ilustración 5.8 y la Ilustración 5.9 podemos obtener la conclusión de que el paquete a través del cual se produce la comunicación entre las clases de *Clients* y las de *Validator* y *Component* es *ejb*, donde ha sido necesario crearse una nueva interfaz.

6 IMPLEMENTACIÓN

Una vez concluidas las fases de análisis y diseño se pasa la fase de implementación, que será donde se recojan los productos obtenidos en etapas anteriores y se desarrolle la solución mediante las diversas tecnologías explicadas en el capítulo 2. El propósito de este apartado es el de dar una visión general del funcionamiento de cada clase creada.

Como se indica en el capítulo anterior, los cambios introducidos en el código fuente de CopperCore se realizan en tres paquetes ya existentes: *Clients*, *Component* y *Validator*.

6.1 PAQUETE CLIENTS

Como ya se ha indicado en la fase de diseño, el paquete *Clients* incluye la implementación de las diferentes interfaces que ofrece CopperCore. Toda esta implementación se ha desarrollado en una única clase, llamada *Clicc*, que es con la que el usuario interactúa a través del intérprete de comandos.

Para añadir el nuevo servicio que permita la publicación de *Adaptation Pokes* se han tenido que modificar una serie de variables globales y crear un método, llamado *uploadPoke()*, de forma que se permita, a través del intérprete de comandos, subir un *Adaptation Poke* y ejecutarlo.

Para ello se comprueba que el nombre del *Adaptation Poke* se corresponde con uno existente, creándose en tal caso un objeto de tipo MyPokePackage (ver apartado 6.2) para encapsular toda la información contenida dentro del archivo zip que representa el *Adaptation Poke* (manifiesto, comandos a ejecutar, etc.). Acto seguido, se valida a través del método *validate()* de la clase MyPokePacakage, desempaquetando los archivos contenidos en el archivo zip en caso de que la validación sea correcta. Por último, se llama, a través de la interfaz *processAdaptation()* del paquete *ejb*, al método de la clase MyAdaptor (ver apartado 6.3) que inicia el proceso de adaptación, *process()*.

6.2 PAQUETE VALIDATOR

El paquete *validator* se encarga de validar cualquier componente de un *Learning Design* incluido en una UOL. Está compuesto por 34 clases, 3 de las cuales han sido añadidas para la realización del módulo de adaptación. A continuación se menciona cada una de ellas y sus principales funcionalidades.

- Clase MyAdaptorData. Se encarga de encapsular los datos que contiene el archivo zip que representa a un *Adaptation Poke*, es decir, el manifiesto y los comandos a ejecutar. La información se almacena en arrays de bytes.

- Clase MyPokePackage. Se encarga de validar un paquete y analizarlo, desempaquetando el contenido del fichero zip que representa el *Adaptation Poke* en caso de que la validación sea correcta. En tal caso, se crea un objeto de tipo MyPokeManifest, donde se almacena el manifiesto incluido en el *Adaptation*

Poke, en caso de que lleve, y un array de bytes para almacenar los comandos con las adaptaciones a realizar.

- Clase MyPokeManifest. Se encarga de almacenar la información que contiene el manifiesto del IMS *Learning Design* que se va a cambiar.

6.3 PAQUETE COMPONENT

Este paquete contiene los diferentes elementos de que consta IMS *Learning Design*. Está formado por 71 clases que implementan la lógica de negocio que ofrece CopperCore, 5 de las cuales han sido creadas para el desarrollo del módulo de adaptación. La información que manejan estas clases está relacionada con actividades, actos, propiedades, *environments*, roles, etc. Ya que es en este paquete donde se maneja toda esta información, se ha creído conveniente añadir aquí las clases que implementan las nuevas funcionalidades requeridas, pues las modificaciones a realizar actúan sobre los diferentes elementos de *Learning Design* aquí contenidos. La relación de clases creadas es la siguiente, incluyendo una descripción de la funcionalidad de cada una de ellas:

- Clase MyAdaptor. Gestiona las adaptaciones indicadas en un *Adaptation Poke*. El proceso es el siguiente:

  - ✓ Procesamiento inicial. Se crea un objeto *MyParserXMLComMan*, al que se le pasan los comandos que contiene el objeto *MyAdaptorData* comentado en el apartado 6.2. Estos comandos son analizados y almacenados en un Vector, listos para ser usados. Acto seguido, se crea un objeto de tipo *MyParserXMLManifest*, al que se le pasa el manifiesto de la UOL que se quiere modificar. Este manifiesto es analizado y preparado para tratar con la información que contiene, la cual se almacena en otro objeto, de tipo *MyPMD*. A continuación, se obtienen el nodo raíz del árbol que representa la UOL y el del árbol que representa a las actividades que contiene esa UOL. Ahora ya estamos preparados para procesar las adaptaciones.

  - ✓ Adaptación del árbol. Una vez que se tiene la raíz del árbol, hay que mirar qué elemento del mismo va a ser modificado. Se pueden dar tres casos:

    - learning-activity. En este caso se pueden dar diferentes casos de acciones a realizar: modificar el estado de completitud o de visibilidad de la actividad, modificar el título de una actividad, modificar el recurso al que apunta una actividad y modificar, borrar o añadir un *environment*. Estas acciones están descritas más abajo.

- activity-structure. En este caso las acciones que se pueden llevar a cabo son: modificar el título de la actividad, modificar el tipo de estructura, modificar, borrar o añadir un *environment*, y añadir o borrar tanto una *learning-activity* como una *support-activity*. Estas acciones están descritas más abajo.

- support-activity. En este caso las acciones que se pueden llevar a cabo son las mismas que en el caso de una *learning-activity*: modificar el estado de completitud o de visibilidad de la actividad, modificar el título de una actividad, modificar el recurso al que apunta una actividad y modificar, borrar o añadir un *environment*. Estas acciones están descritas más abajo.

✓ Modificar el estado de completitud de una *learning-activity* o de una *support-activity*. Se busca el nodo correspondiente en el árbol a través del identificador obtenido del *Adaptation Poke* y se le cambia el valor al atributo de esa actividad.

✓ Modificar el estado de visibilidad de una *learning-activity* o de una *support-activity*. Se busca el nodo correspondiente en el árbol a través del identificador obtenido del *Adaptation Poke* y se le cambia el valor al atributo de esa actividad.

✓ Modificar el título o recurso de una *learning-activity* y de una *support-activity* o el título o el tipo de estructura de una *activity-structure*. Se busca el nodo correspondiente en el árbol a través del identificador obtenido del *Adaptation Poke*, se navega por él hasta encontrar el elemento que contiene el atributo que se desea cambiar y se le asigna el valor pertinente. Tras esto hay que hacer los cambios definitivos en la UOL, para lo que se usará el método *persist()* sobre el objeto que contiene al nodo del árbol.

✓ Modificar el *environment* de una *learning-activity*, de una *support-activity* o de una *activity-structure*. Se busca el nodo correspondiente en el árbol a través del identificador obtenido del *Adaptation Poke* y se cambia el atributo *environment*. Después hay que modificar una serie de objetos, de tipo *PropertyDefDto* y *ActivityTreePropertyDef*, tras lo que se hace un *persist()* para que los cambios se guarden en la UOL.

✓ Añadir un *environment* a una *learning-activity*, a una *support-activity* o a una *activity-structure*. Lo primero que hay que hacer es buscar el *environment* que se quiere añadir en el objeto *MyPMD* que lo contiene. Este objeto *MyPMD* almacena toda la información del nuevo manifiesto

de la UOL, incluido en el *Adaptation Poke*. Acto seguido, se crea un objeto de tipo *MyXMLConverter*, al que se le pasa el *environment* encontrado anteriormente y el objeto *MyPMD*. Este objeto *MyXMLConverter* construirá un archivo XML con la información obtenida del manifiesto del *Adaptation Poke*, de forma que se obtenga el formato que necesita CopperCore. A continuación hay dos posibilidades: que el entorno a añadir sea de tipo *service* o de tipo *learning-object*. En el primer caso, se crean objetos de tipo *LearningObjectPropertyDef* y *LearningObjectProperty*, que contienen la información del nuevo *learning-object*, y desde los que se llama al método *persist()* para hacer los cambios definitivos. En el segundo caso, los objetos son de tipo *MonitorPropertyDef* y *MonitorProperty*, y también hay que llamar al método *persist()*. Por último, se tienen que crear otro tipo de objetos para dejar el árbol consistente, como son *EnvironmentPropertyDef* y *EnvironmentProperty*, para el nuevo *environment* que se va a añadir, y *EnvironmentTreePropertyDef* y *EnvironmentTreeProperty*, para el árbol que contiene los *environments*. Se hace *persist()* con todos estos objetos para que el cambio sea definitivo. Una vez hechos todos estos pasos, es necesario navegar por el árbol de actividades hasta encontrar el nodo que va a hacer referencia al nuevo entorno añadido, donde habrá que buscar su atributo *environment* y actualizarlo.

- ✓ Borrar un *environment* de una *learning-activity*, de una *support-activity* o de una *activity-structure*. El primer paso consiste en crear un objeto de tipo *EnvironmentTreeProperty*, que contendrá el árbol de *environments* que alberga el *environment* que se quiere eliminar. Una vez borrado el nodo del árbol, se crea un objeto de tipo *EnvironmentTreePropertyDef*, que contendrá el nuevo árbol de *environments* de la actividad referenciada en el *Adaptation Poke*.

- ✓ Añadir una *learning-activity* o una *support-activity* a una *activity-structure*. El primer paso es obtener el nodo de la nueva actividad del manifiesto del *Adaptation Poke*. Acto seguido, se crea un objeto de tipo *MyXMLConverter*, que convertirá el nodo de la actividad obtenido en un archivo con formato XML. Si se trata de añadir una *learning-activity*, se crearán los dos tipos de objetos que representan este tipo de actividad en CopperCore, *LearningActivityProperty* y *LearningActivityPropertyDef*, desde los que se llamará al método *persist()* para hacer los cambios permanentes. Si por el contrario se trata de añadir una *support-activity*, los objetos que se crearán serán de tipo *SupportActivityProperty* y *SupportActivityPropertyDef*, que también llamarán a sus métodos *persist()*. A continuación se añade el nuevo nodo al árbol de actividades en el lugar que le corresponda, partiendo de la información que contiene el objeto *MyXMLConverter* anteriormente creado.

- ✓ Borrar una *learning-activity* o una *support-activity* a una *activity-structure*. El primer paso es obtener el nodo del árbol de actividades que contiene la *activity-structure* a través del identificador obtenido del *Adaptation Poke*. Una vez obtenido, se busca entre sus hijos la actividad a borrar, y se elimina del árbol. Se hace *persist()* para que el cambio sea permanente.

- Clase MyParserXMLComMan. Se encarga de leer un *Adaptation Poke* y extraer las adaptaciones que se tienen que llevar a cabo. Para ello, parsea el archivo poke.xml que contiene el *Adaptation Poke*. A través del identificador de la acción, se distinguen tres tipos de acciones a realizar:

  - ✓ Inserción. Se crea un array de *String* que contiene la siguiente información:

    - Acción a realizar. En este caso, *add*.
    - Identificador de la actividad donde se va a realizar la inserción.
    - Identificador del elemento a añadir.

  - ✓ Borrado. Se crea un array de *String* que contiene la siguiente información:

    - Acción a realizar. En este caso, *del*.
    - Identificador de la actividad donde se va a realizar el borrado.
    - Identificador del elemento a borrar.

  - ✓ Modificación. Se crea un array de *String* que contiene la siguiente información:

    - Acción a realizar. En este caso, *upd*.
    - Identificador del elemento a cambiar (actividad, *environment*).
    - Nombre del atributo que se desea cambiar.
    - Nuevo valor del atributo.

- Clase MyParserXMLManifest. Se encarga de leer el manifiesto de un *Adaptation Poke* y procesarlo. Lo primero que hace es obtener el nodo raíz del manifiesto, tras lo cual empieza a parsearlo. Va leyendo nodo a nodo y almacena en un objeto de tipo *MyPMD* la información de todos los elementos que componen un *Learning Design*. Cada nodo puede contener diferentes elementos, por lo que existen numerosos métodos para tratar todos los casos.

- Clase MyPMD. Se encarga de manejar los elementos de un manifiesto, almacenando en diferentes objetos cada uno de ellos (metadatos, organizaciones, recursos, roles, actividades, propiedades, *environments*, etc.).

- Clase MyXMLConverter. Se encarga de transformar la información de un objeto de tipo *MyPMD*, que contiene la información del manifiesto de un *Adaptation Poke*. Existen tres posibles casos en los que son necesarias estas transformaciones:

  - ✓ Insertar una nueva actividad. En este caso, se recorre la parte del árbol del manifiesto del *Adaptation Poke* que hace referencia a la nueva actividad, transformando cada elemento encontrado a XML y almacenándolo en un objeto de tipo *String*. Este *String* es devuelto en el formato que necesita CopperCore internamente para añadir nuevos elementos.

  - ✓ Insertar un *learning-object* en un *environment*. En este caso, se recorre la parte del árbol del manifiesto del *Adaptation Poke* que hace referencia a al nuevo *learning-object*, transformando cada elemento encontrado a XML y almacenándolo en un objeto de tipo *String*. Este *String* es devuelto en el formato que necesita CopperCore internamente para añadir nuevos elementos.

  - ✓ Insertar un *service* en un *environment*. En este caso, se recorre la parte del árbol del manifiesto del *Adaptation Poke* que hace referencia al nuevo *service*, transformando cada elemento encontrado a XML y almacenándolo en un objeto de tipo *String*. Este *String* es devuelto en el formato que necesita CopperCore internamente para añadir nuevos elementos.

7 MANUAL DE USUARIO

En esta sección se va a describir el entorno en el que se ha desarrollado este Proyecto Fin de Carrera, de tal forma que cualquier persona que lea esta sección sea capaz de usar tanto CopperCore como el módulo de adaptación desarrollado.

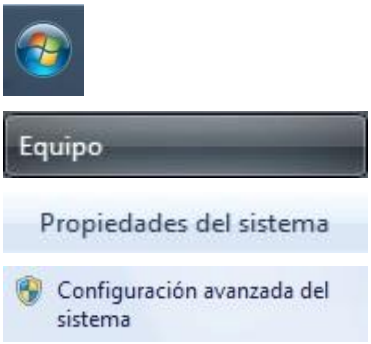
7.1 CONFIGURACIÓN DEL ENTORNO

Si se quiere utilizar CopperCore sin el módulo de adaptación, hay que descargarse el entorno de ejecución de CopperCore (CCRT, de sus siglas en inglés *CopperCore Run Time environment*) [25] versión 3.2. Una vez descargado, se descomprime y ya está listo para usar.

En el caso de querer usar el módulo de adaptación, hay que usar el CCRT que se encuentra en la carpeta *modulo\ccrt*, disponible en el CD adjunto a esta memoria.

Para usar el CCRT es necesario tener instalada la versión del J2EE SDK 1.4.2 [26]. Una vez realizada la instalación, es necesario configurar la variable de entorno JAVA_HOME, que debe apuntar al directorio de instalación de Java. Para ello basta con:

- Iniciar Windows (7 o Vista).
- Seleccionar equipo.
- Pulsar sobre Propiedades del sistema.
- Pulsar sobre Configuración avanzada del sistema.



Tras esto aparecerá la pantalla Propiedades del sistema, en la que habrá que seleccionar Variables de entorno. Se puede ver en la Ilustración 7.1.

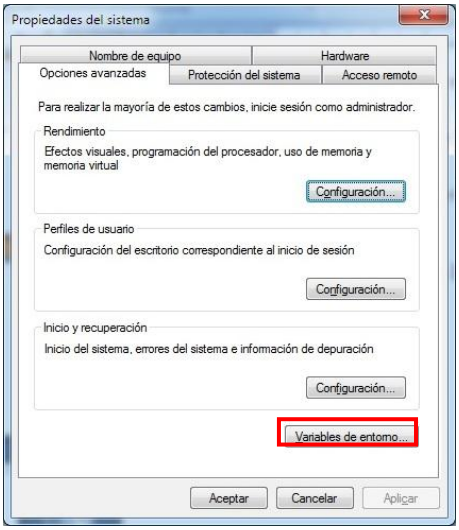


Ilustración 7.1 Pantalla de Propiedades del sistema

Tras pulsar el botón variables de entorno aparecerá una última pantalla, donde se añadirá una nueva Variable del sistema pulsando sobre Nueva y escribiendo JAVA_HOME como nombre y la ruta de la carpeta de instalación de Java como valor (por ejemplo, C:\Program Files\Java\jdk1.4.2_19). Se puede apreciar el proceso en la Ilustración 7.2.

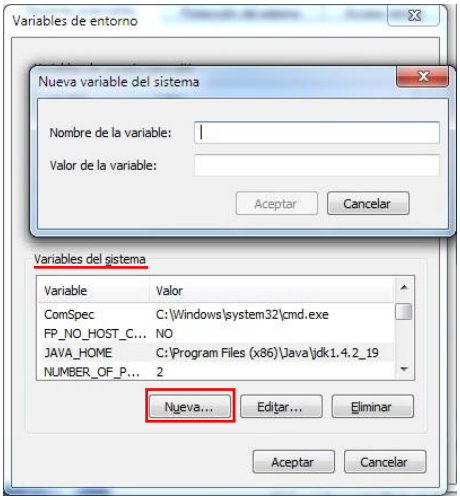


Ilustración 7.2 Pantalla de Variables de entorno

Para poder compilar el código fuente es necesario tener instalado Ant versión 1.6 [20]. Una vez desempaquetado el archivo zip en un directorio local, procurando no dejar espacios en blanco en la ruta, es necesario configurar la variable de entorno ANT_HOME. Para hacerlo basta con seguir los pasos anteriores, y seleccionar como ruta el directorio donde se ha descomprimido Ant.

Para poder utilizar el reproductor de CopperCore es necesario tener instalado el navegador Microsoft Internet Explorer 6 o superior o Mozilla Firefox, aunque en este último pueden producirse fallos.

7.2 EJECUCIÓN DE COPPERCORE

Una vez realizados los pasos anteriores ya se puede ejecutar la aplicación. Para ello se han creado unos *script*, para que sea un proceso más llevadero. Lo primero es abrir una ventana de comandos (Windows + r, y teclear cmd). Tras ello, navegar por el CD hasta la carpeta *modulo\CopperCore*. Una vez en este directorio, se puede teclear lo siguiente:

- *build*: sirve para compilar. Se lanza Ant, que lee el archivo *build.xml* y realiza la compilación con las librerías que se indican, generando los archivos *coppercore.ear*, *coppercore.sar*, *publisher.war*, *coppercore.jar*, *coppercore-client.jar* y *coppercore-common.jar* y la documentación javadoc, contenida en la carpeta *modulo\CopperCore\doc*. Las rutas que apuntan a las diferentes librerías en los archivos *build.bat* y *build.xml* son locales.

- *ejecutar*: sirve para lanzar la aplicación. Se copian los archivos *coppercore.ear*, *coppercore-client.jar* y *coppercore-common* a diferentes directorios de *modulo\ccrt*. Después se ejecutan los archivos de la carpeta *modulo\ccrt coppercore.bat*, que lanza el CCRT, y *clicc.bat*, que lanza el intérprete de comandos. Si se producen problemas con la variable de entorno JAVA_HOME, hay que eliminarla, primero seleccionándola y después pulsando el botón Eliminar de la pantalla mostrada en la Ilustración 7.2.

Tras escribir *ejecutar* se pedirán 3 confirmaciones de escritura, y se escribirá las 3 veces que sí. Una vez realizado, se abre una nueva ventana, llamada *coppercore.bat*, en la que tras pulsar espacio se lanza el CCRT (Ilustración 7.3). Tras esto, la ventana *coppercore.bat* se puede cerrar. Una vez cargado (Ilustración 7.4, donde se aprecia la indicación *Started in XXs:XXXms*), esta ventana se deja abierta y en la ventana *ejecutar* se vuelve a pulsar espacio (pide presionar una tecla para continuar). Tras ello se lanzará la consola de gestión de CopperCore, también conocida como herramienta Clicc, de sus siglas en inglés *Command Line Interface CopperCore*, abierta en una ventana con el nombre *clicc.bat*, con la que podremos hacer todas las gestiones de CopperCore (Ilustración 7.5). Para dejar de usar el CCRT y Clicc basta con cerrar sus respectivas ventanas.

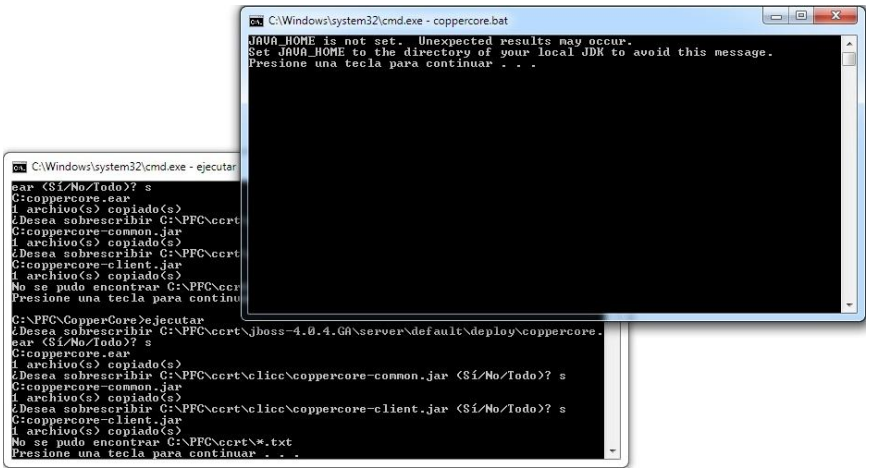


Ilustración 7.3 Pantalla ejecutar

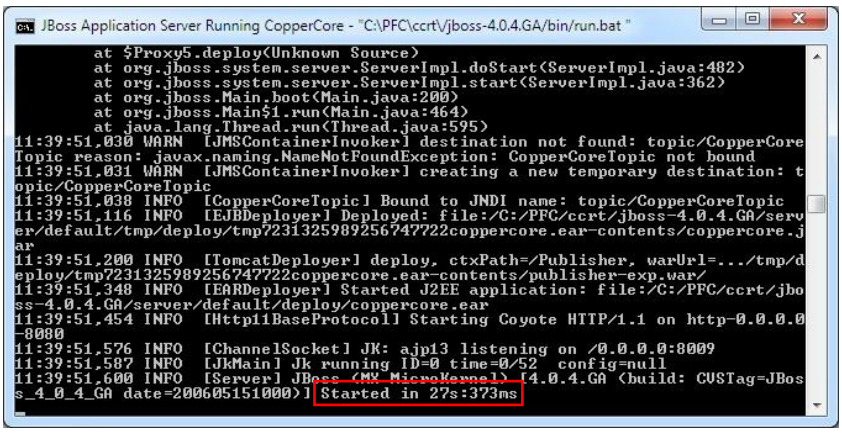


Ilustración 7.4 Pantalla JBoss Application Server Running CopperCore

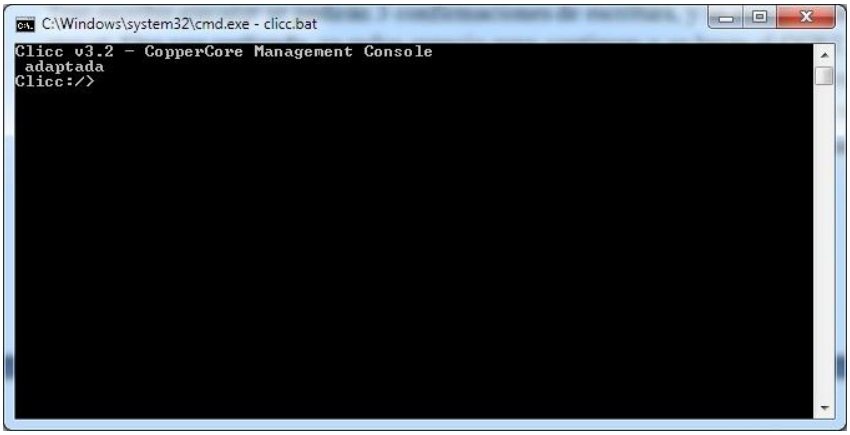


Ilustración 7.5 Pantalla de la consola de gestión de CopperCore

Una vez lanzado el CCRT se puede publicar una UOL en CopperCore. Para eso hay que abrir un navegador web y escribir la siguiente dirección:

http://localhost:8080/Publisher/publication.html

Tendremos la interfaz que se muestra en la Ilustración 7.6.

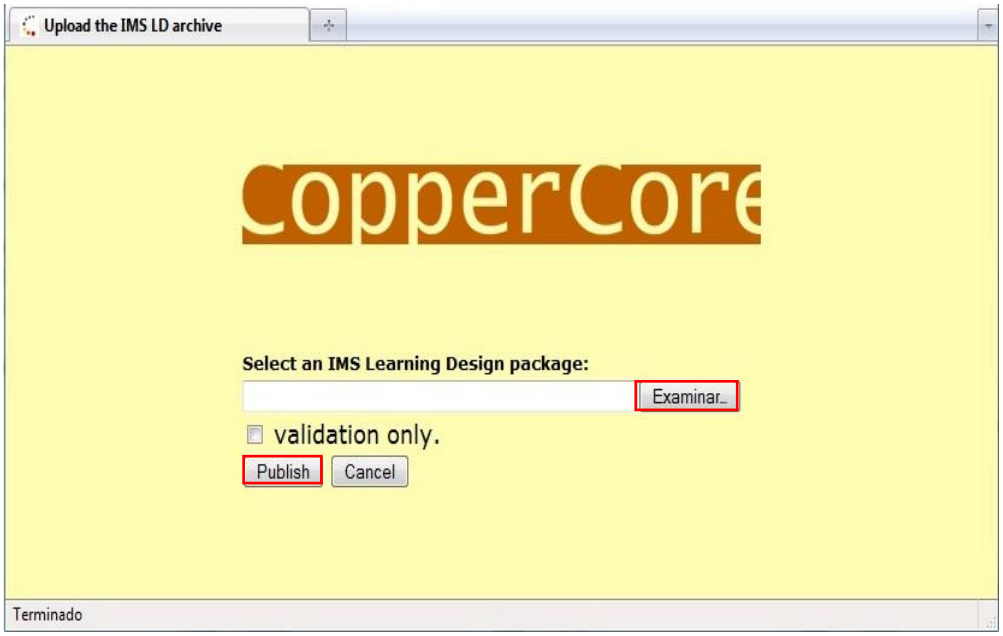


Ilustración 7.6 Pantalla Publisher

A través de esta interfaz podemos publicar una UOL. Para ello hay que pulsar el botón Examinar y seleccionar una. Como ejemplo se puede publicar la que se encuentra dentro del CD en la ruta *modulo\ccrt\CaseStudyPokeCreator.zip*. Una vez seleccionada la UOL, se pulsa el botón Publish. Una publicada la UOL, comenzará el proceso de validación, por lo que podremos ver la pantalla mostrada en la Ilustración 7.7. Si hubiera algún problema con la UOL, se mostraría.

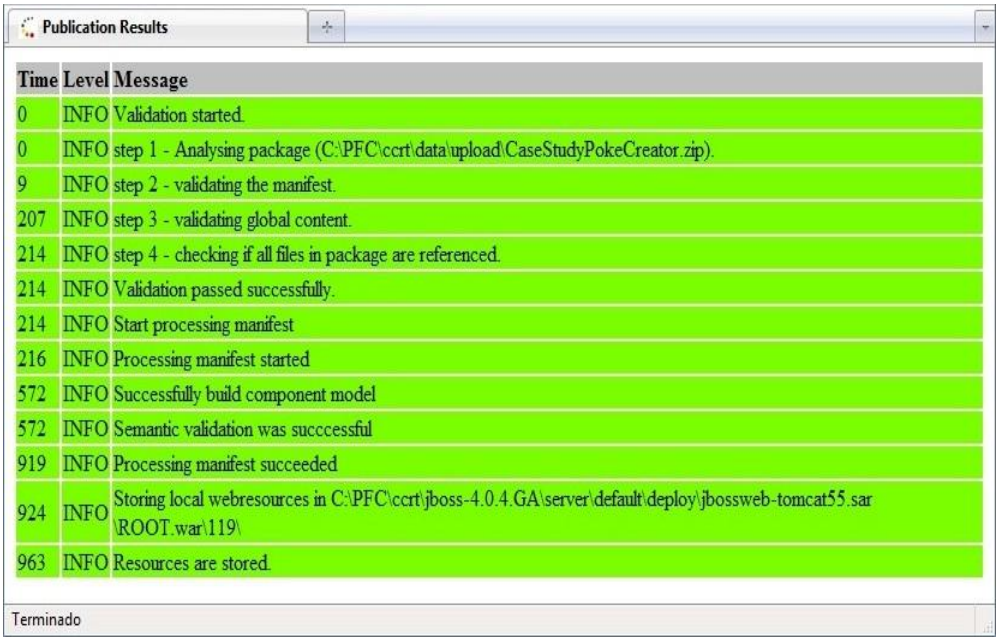


Ilustración 7.7 Pantalla de validación

Una vez que ya tengamos la UOL publicada y correctamente validada, podemos usar el reproductor de CopperCore abriendo un navegador web y escribiendo la dirección:

http://localhost:8080/WebPlayer/runswitch.html

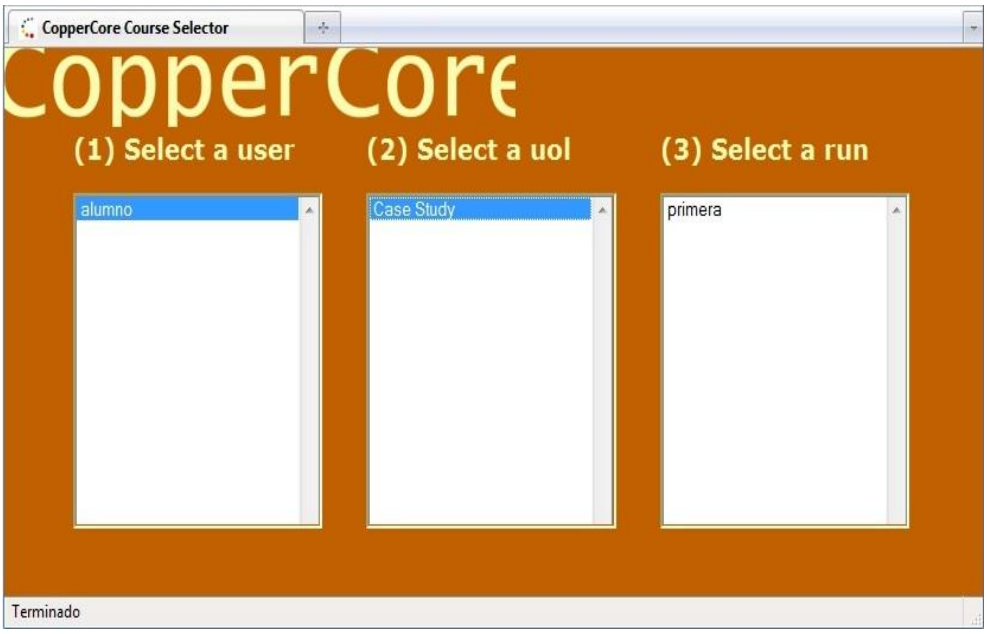


Ilustración 7.8 Pantalla del reproductor

En la Ilustración 7.8 podemos ver la interfaz del reproductor. Su uso es muy sencillo: en primer lugar hay que seleccionar un usuario, previamente creado (ver capítulo siguiente); en segundo lugar se selecciona una UOL publicada; y por último se escoge una ejecución, también previamente creada (ver capítulo siguiente). Tras ello se abre

una nueva ventana en el navegador en la que se muestra la interfaz que figura en la Ilustración 7.9, en la que se puede ver la UOL desplegada.

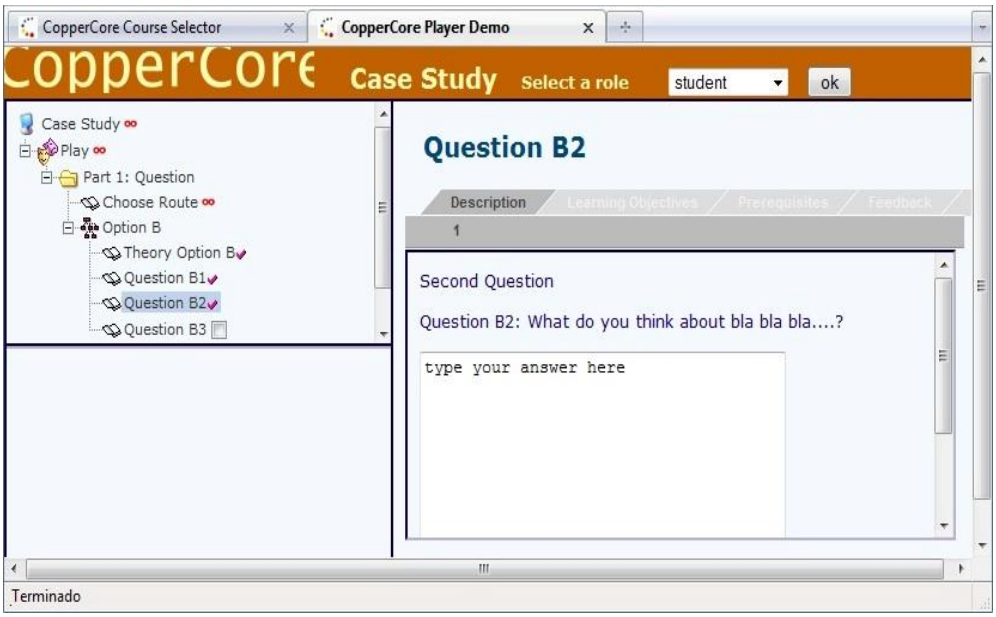


Ilustración 7.9 Pantalla con la UOL desplegada

Como se puede ver en la Ilustración 7.9, hay tres paneles principales en la pantalla. El panel superior izquierdo contiene el árbol de la actividad para el usuario seleccionado y para su rol. Cada vez que se selecciona un nodo en este árbol, se rellena el árbol de *environments* con la jerarquía de los elementos del *environment* asociado a dicha actividad. El panel derecho muestra siempre el contenido del nodo de la actividad o del *environment* seleccionado. Este panel de contenido puede contener una o más pestañas que pueden estar de un color más claro si no están disponibles. Debajo de estas pestañas se muestran uno o más números de página que representan los ítems de cada tópico. Estas páginas pueden ser seleccionadas con la simple acción de pinchar sobre los números.

7.3 CONSOLA DE GESTIÓN DE COPPERCORE

La consola de gestión de CopperCore también recibe el nombre de Clicc, y se puede ver en la Ilustración 7.5. Su funcionamiento está basado en los comandos. Cada vez que se introduce uno de ellos es necesario pulsar la tecla Enter. Se puede ver una lista con todos los comandos disponibles en la Ilustración 7.10, para lo que bastaría con teclear *help*.

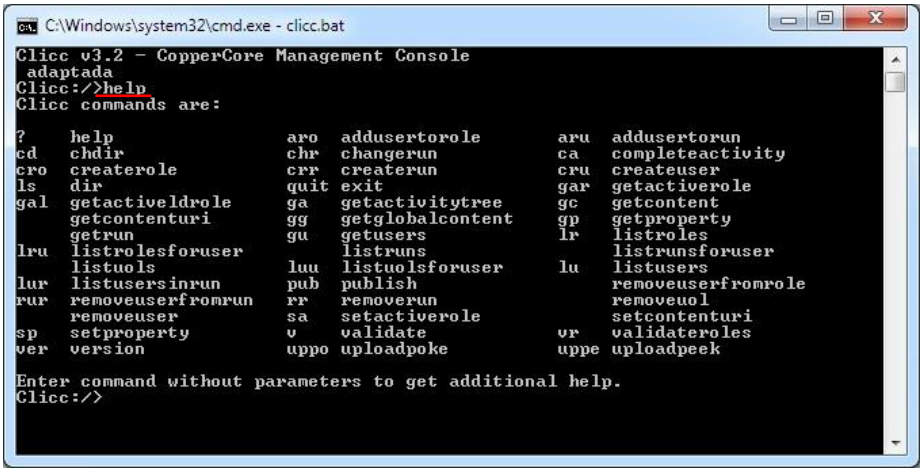


Ilustración 7.10 Pantalla de Clicc con los comandos disponibles

Para poder ver cada UOL publicada a través de la interfaz expuesta en la Ilustración 7.9 es necesario seguir una serie de pasos y operar con un conjunto de comandos de los mostrados en la Ilustración 7.10.

El primer paso consiste en la creación de nuevos usuarios. Es necesario al menos uno. Para ello hay que teclear el comando *createuser <id del usuario>*. En este caso introduciremos *createuser alumno*. Una vez creados los usuarios que se estimen oportunos, se pasa a listar las UOL publicadas mediante la introducción del comando *dir*. Estos dos primeros pasos pueden verse reflejados en la Ilustración 7.11.

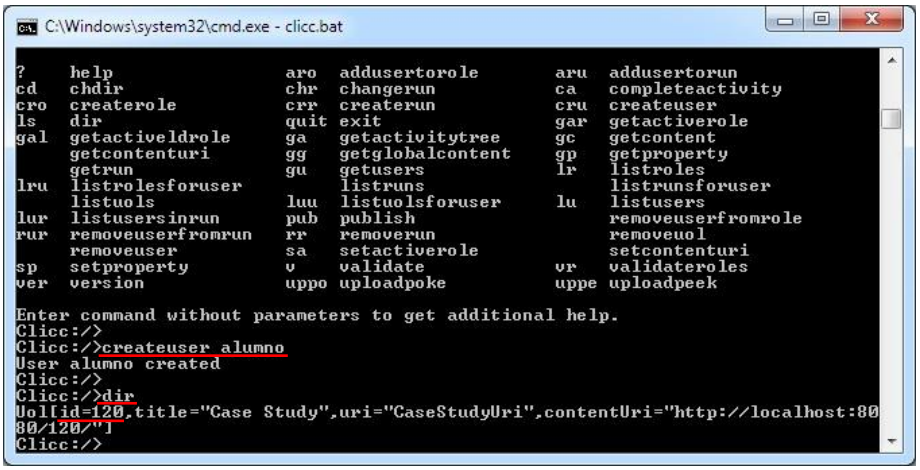


Ilustración 7.11 Pantalla de Clicc con los comandos createuser y dir

Como se puede observar en la Ilustración 7.11, cada UOL viene identificada por un identificador único. Este identificador se usa para situarse en la publicación que se desea. El siguiente paso en el proceso es teclear *cd <id de la UOL>*. En nuestro caso introduciremos *cd 120*. Tras ello, la ventana de comandos cambia su ruta (antes se leía *Clicc:/>*, y ahora se lee *Clicc:/uol=120>*. Esta ruta se actualizará cada vez que se use el comando *cd*.

El próximo paso consiste en la creación de una ejecución. Para ello bastará con introducir el comando *createrun <titulo de la ejecución>*. En nuestro ejemplo teclearemos *createrun primera*. Si se quieren introducir espacios en el título, será necesario que todo el texto vaya incluido entre comillas (*createrun "primera ejecución"*). Para crear una ejecución es necesario haberse situado antes en la UOL donde se quiera crear. También hay que comentar que Clicc responde a este comando devolviendo el identificador de la ejecución, que será necesario en próximos pasos. Esta parte del proceso puede verse en la Ilustración 7.12.

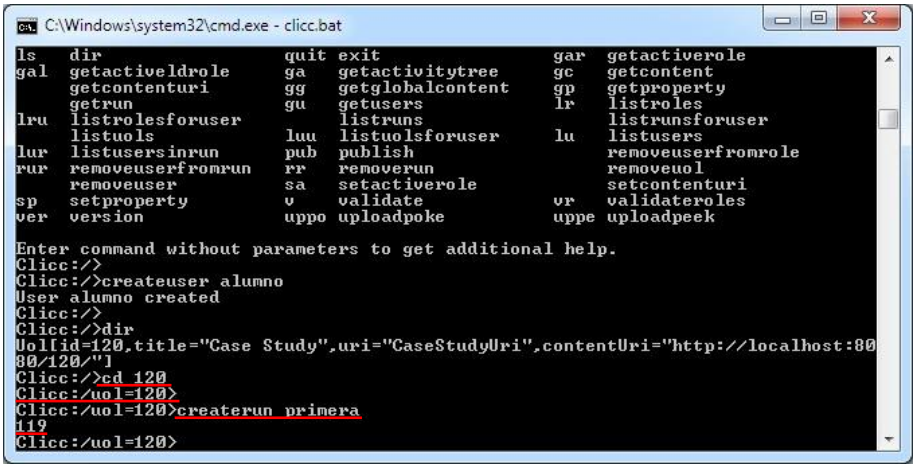


Ilustración 7.12 Pantalla de Clicc con los comandos cd y createrun

La próxima fase del proceso consiste en seleccionar una ejecución, que se hará tecleando *cd <id de la UOL>*, que en nuestro caso será *cd 119*. Tras esto Clicc quedará en la ruta *Clicc:/uol=120/run=119>*. Ahora se añadirán los usuarios que se quieren añadir a esta ejecución. Para ello se introducirá el comando *addusertorun <id del usuario>*. En nuestro ejemplo será *addusertorun alumno*. Una vez añadido, habrá que seleccionarlo con el comando *cd*, como se ha venido haciendo hasta ahora (*cd alumno*), tras lo que Clicc quedará en la ruta *Clicc:/uol=120/run=119/user=alumno>*. Estas acciones se pueden ver reflejadas en la Ilustración 7.13.

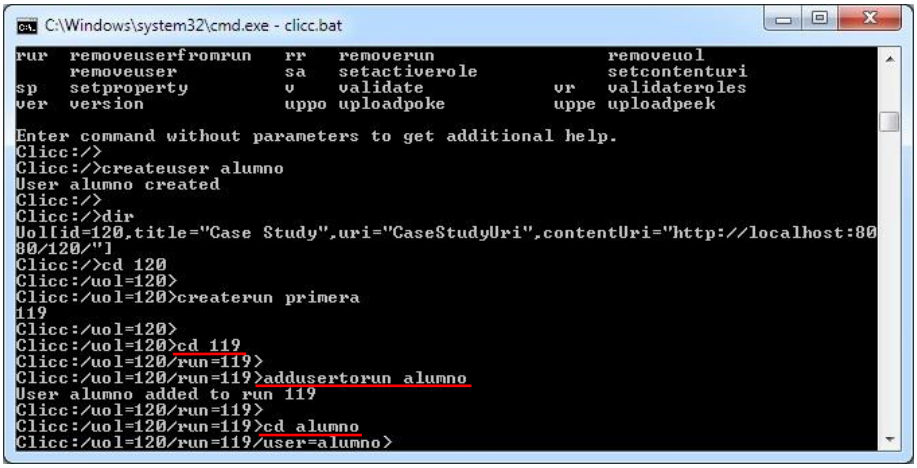


Ilustración 7.13 Pantalla de Clicc con el comando addusertorun

Una vez que se ha seleccionado un usuario para una ejecución, toca asignarle un rol. Lo primero que hay que hacer es listar todos los roles disponibles, que están definidos en la UOL publicada. Esto se consigue introduciendo el comando *listroles*. Clicc devuelve el identificador asociado a cada rol. Tras haber elegido un rol para el usuario, es el turno de hacer la asignación. Teclearemos el comando *addusertorole <id del usuario> <id del rol>*, en nuestro ejemplo *addusertorole alumno 323*, que corresponde con el rol de estudiante. Por último, habrá que activar el rol. Para ello bastará con introducir el comando *setactiverole <id del rol>*, en nuestro caso *setactiverole 323*. Con esto conseguiremos que cuando el navegador web esté lanzado sea la vista del rol estudiante la que se vea. Podemos ver todo este proceso en la Ilustración 7.14.

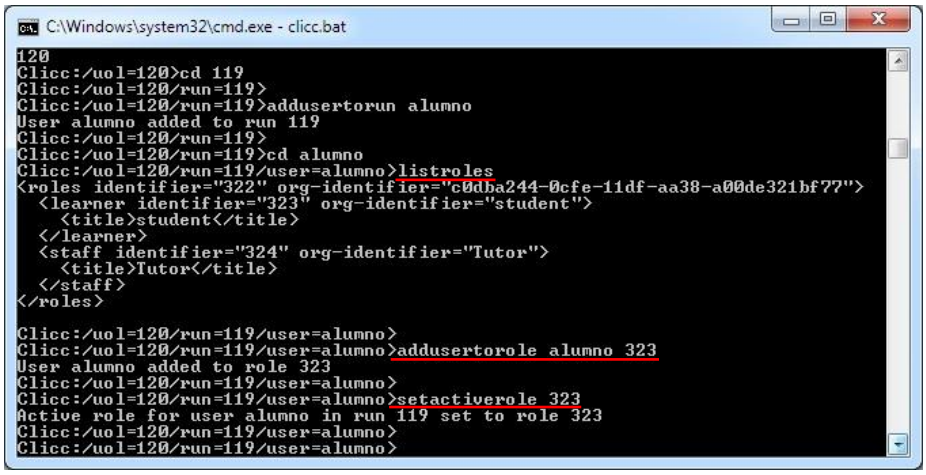


Ilustración 7.14 Pantalla de Clicc con los comandos listroles, addusertorole y setactiverole

Todo el proceso contenido entre los comandos *addusertorun* y *addusertorole* deberá ser repetido cada vez que queramos añadir un usuario a una ejecución y asignarle un rol. Para ello lo primero que habrá que hacer será teclear *cd .. .* Activar un rol sólo deberá hacerse una vez.

Una vez realizados todos los pasos anteriores, ya estamos listos para publicar un *Adaptation Poke*. Los *Adaptation Poke* deberán estar en la carpeta donde se encuentre el

CCRT, que en nuestro caso es *modulo\ccrt*. Se introduce el comando *uploadpoke <id de la UOL> <archivo>*, que en nuestro ejemplo será *uploadpoke 120 pokeUpdLADefaultTitle.zip*. Con este *Adaptation Poke* se pretende cambiar el título de una actividad. Este último paso se puede ver reflejado en la Ilustración 7.15.

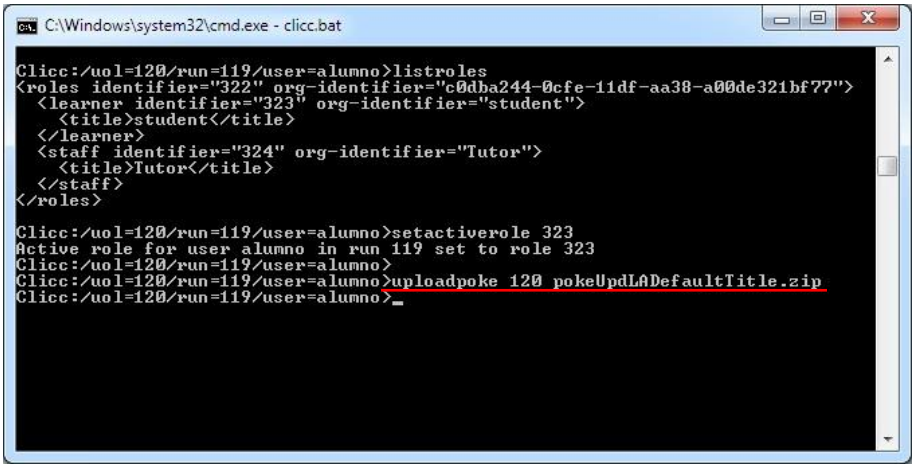


Ilustración 7.15 Pantalla de Clicc con el comando uploadpoke

En la Ilustración 7.16 se puede ver el resultado de aplicar este *Adapatation Poke*. Si nos fijamos en la Ilustración 7.9 podemos apreciar cómo el título antes era *Question B2* y ahora es *Pregunta B2.*

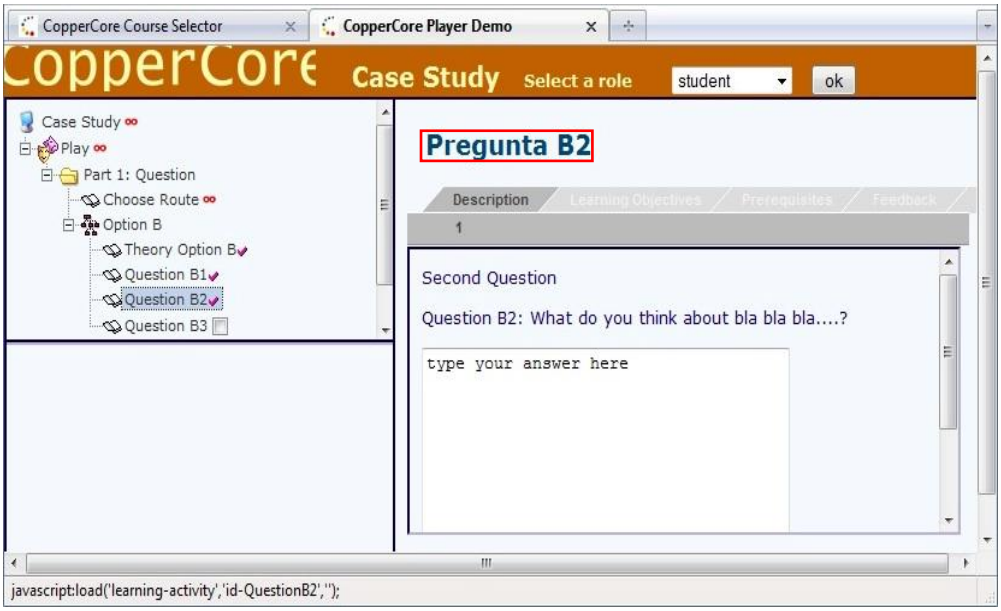


Ilustración 7.16 Pantalla con cambios en la UOL

8 EVALUACIÓN

La finalidad de esta sección es la de demostrar la validez del módulo de adaptación desarrollado, de modo que se verifique que esta solución resuelve los problemas planteados y que cumple los objetivos propuestos, ajustándose de esta manera a los requisitos y funcionalidades exigidas por el cliente.

La evaluación se va a realizar a través de un conjunto de pruebas. El objetivo que se persigue es comprobar si la salida obtenida se corresponde con la esperada, de manera que se pueda verificar el correcto funcionamiento del sistema.

8.1 PLAN DE PRUEBAS

Las diferentes pruebas realizadas sobre el módulo de adaptación desarrollado persiguen un fin: comprobar que los requisitos exigidos por el cliente se cumplen. Esta verificación se ha llevado a cabo de manera que se ha realizado al menos una prueba por cada uno de los requisitos funcionales expuestos en la fase de análisis, lo que permitirá realizar más adelante una matriz de trazabilidad.

Cada uno de los casos de pruebas ejecutados posee las siguientes características:

- Identificador. Este campo facilitará la posterior trazabilidad. Estará compuesto de la siguiente manera: CP-<número>, donde <número> será una cifra de tres dígitos, comenzando por 001.

- Propósito. Se detalla el fin perseguido por la prueba.

- Requisitos vinculados. Indica el requisito funcional de donde surge la prueba. Facilita la posterior trazabilidad.

- Dependencias. Indica si hay una prueba que debe ejecutarse son anterioridad.

- Entrada. Indica los datos necesarios para poder realizar la prueba.

- Salida. Indica el resultado que se desea obtener.

- Resultado. Indica si se ha obtenido la salida que se deseaba. Puede tomar los valores OK o fallo.

- Fecha. Indica la fecha en que se realizó la prueba.

- Archivo. Indica el nombre del archivo donde se encuentra la prueba (*Adaptation Poke*).

- Fallo producido. Indica el resultado erróneo que ha tenido lugar, si procede.

Para realizar las pruebas se ha publicado la UOL *CaseStudyPokeCreator*, disponible en *modulo\ccrt*. Se han creado 2 usuarios, tutor y alumno, con los roles *Tutor* y *student* respectivamente. Por último, se han subido cada uno de los *Adaptation Poke* indicados en el atributo archivo de cada caso de prueba. En total se han implementado 36 *Adaptation Poke*, disponibles en *modulo\ccrt\Adaptation Pokes*.

Los casos de prueba se van a dividir en secciones, atendiendo al criterio del tipo de adaptación que se realiza (modificación, adición o borrado).

8.1.1 CASOS DE PRUEBA DE MODIFICACIÓN

CP-001			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLAEnvironment.zip
Resultado	Fallo	Fecha	16/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* cambia su *environment*. Se queda vacío. Inicialmente tenía dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	No se ven los cambios en el reproductor.		

Tabla 8.1 Caso de prueba CP-001

CP-002			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-001	Archivo	pokeUpdLAEnvironment.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* cambia su *environment*. Se queda vacío. Inicialmente tenía dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.2 Caso de prueba CP-002

CP-003			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLAEnvironment2.zip
Resultado	Fallo	Fecha	16/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su *environment*. Se queda con un *service*. Inicialmente vacío. El cambio es apreciable en el reproductor.		
Fallo producido	No se ven los cambios en el reproductor.		

Tabla 8.3 Caso de prueba CP-003

CP-004			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-003	Archivo	pokeUpdLAEnvironment2.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su *environment*. Se queda con un *service*. Inicialmente vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.4 Caso de prueba CP-004

CP-005			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLAEnvironment3.zip
Resultado	Fallo	Fecha	21/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su *environment*. Se queda con dos *service*. Inicialmente vacío. El cambio es apreciable en el reproductor.		
Fallo producido	No se ven los cambios en el reproductor.		

Tabla 8.5 Caso de prueba CP-005

CP-006			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-005	Archivo	pokeUpdLAEnvironment3.zip
Resultado	Fallo	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su *environment*. Se queda con dos *service*. Inicialmente vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.6 Caso de prueba CP-006

CP-007			
Propósito	Cambiar el *environment* de una *learning-activity*.		
Requisitos vinculados	RF-001, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLAEnvironment4.zip
Resultado	OK	Fecha	21/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* cambia su *environment*. Se queda con un *service*. Inicialmente tiene dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.7 Caso de prueba CP-007

CP-008			
Propósito	Cambiar el *environment* de una *support-activity*.		
Requisitos vinculados	RF-002, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSAEnvironment.zip
Resultado	OK	Fecha	21/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* cambia su *environment*. Se queda vacío. Inicialmente tiene un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.8 Caso de prueba CP-008

CP-009			
Propósito	Cambiar el *environment* de una *support-activity*.		
Requisitos vinculados	RF-002, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSAEnvironment2.zip
Resultado	OK	Fecha	21/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* cambia su *environment*. Se queda con dos *service*. Inicialmente tiene un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.9 Caso de prueba CP-009

CP-010			
Propósito	Cambiar el *environment* de una *activity-structure*.		
Requisitos vinculados	RF-003, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdASEnvironment.zip
Resultado	Fallo	Fecha	16/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* cambia su *environment*. Se queda con un *service*. Inicialmente está vacío. El cambio es apreciable en el reproductor.		
Fallo producido	Fallo al intentar leer el archivo imsmanifest.xml.		

Tabla 8.10 Caso de prueba CP-010

CP-011			
Propósito	Cambiar el *environment* de una *activity-structure*.		
Requisitos vinculados	RF-003, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-010	Archivo	pokeUpdASEnvironment.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* cambia su *environment*. Se queda con un *service*. Inicialmente está vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.11 Caso de prueba CP-011

CP-012			
Propósito	Cambiar el *environment* de una *activity-structure*.		
Requisitos vinculados	RF-003, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdASEnvironment2.zip
Resultado	OK	Fecha	21/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* cambia su *environment*. Se queda con dos *service*. Inicialmente está vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.12 Caso de prueba CP-012

CP-013			
Propósito	Cambiar el título de una *activity-structure*.		
Requisitos vinculados	RF-004, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdASDefaultTitle.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* cambia su título. Pasa de *"Discuss about the questions"* a "Discusiones sobre las respuestas". El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.13 Caso de prueba CP-013

CP-014			
Propósito	Cambiar el tipo de una *activity-structure*.		
Requisitos vinculados	RF-005, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdASDefaultType.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* cambia su tipo. Pasa de *sequence* a *selection*. El cambio es apreciable con trazas en el código.		
Fallo producido	-		

Tabla 8.14 Caso de prueba CP-014

CP-015			
Propósito	Cambiar el título de una *learning-activity*.		
Requisitos vinculados	RF-006, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLADefaultTitle.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su título. Pasa de *"Question B2"* a *"Pregunta B2"*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.15 Caso de prueba CP-015

CP-016			
Propósito	Cambiar el título de una *support-activity*.		
Requisitos vinculados	RF-007, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSADefaultTitle.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* cambia su título. Pasa de *"Monitor the learners' responses"* a *"Cambio realizado en el título"*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.16 Caso de prueba CP-016

CP-017			
Propósito	Cambiar la referencia al recurso utilizado en una *learning-activity*.		
Requisitos vinculados	RF-008, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLADefaultURI.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su recurso. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.17 Caso de prueba CP-017

CP-018			
Propósito	Cambiar la referencia al recurso utilizado en una *support-activity*.		
Requisitos vinculados	RF-009, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSADefaultURI.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* cambia su recurso. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.18 Caso de prueba CP-018

CP-019			
Propósito	Cambiar el estado de visibilidad de una *learning-activity*.		
Requisitos vinculados	RF-010, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLAVisibility.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* cambia su estado de visibilidad. Pasa de *true* a *false*. El cambio es apreciable mediante trazas en el código.		
Fallo producido	-		

Tabla 8.19 Caso de prueba CP-019

CP-020			
Propósito	Cambiar el estado de visibilidad de una *support-activity*.		
Requisitos vinculados	RF-011, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSAVisibility.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* cambia su estado de visibilidad. Pasa de *true* a *false*. El cambio es apreciable mediante trazas en el código.		
Fallo producido	-		

Tabla 8.20 Caso de prueba CP-020

CP-021			
Propósito	Completar una *learning-activity*.		
Requisitos vinculados	RF-012, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdLACompletion.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno. Es necesario que la actividad no esté completada.		
Salida	La *learning-activity* se completa. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.21 Caso de prueba CP-021

CP-022			
Propósito	Completar una *support-activity*.		
Requisitos vinculados	RF-013, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeUpdSACompletion.zip
Resultado	OK	Fecha	24/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor. Es necesario que la actividad no esté completada.		
Salida	La *support-activity* se completa. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.22 Caso de prueba CP-022

8.1.2 CASOS DE PRUEBA DE ADICIÓN

CP-023			
Propósito	Añadir un *learning-object* al *environment* de una *learning-activity*.		
Requisitos vinculados	RF-014, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddLAEnvironment.zip
Resultado	Fallo	Fecha	15/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* actualiza su *environment*. Se queda con dos *service* y un *learning-object*. Inicialmente tiene dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	No se ve en el reproductor.		

Tabla 8.23 Caso de prueba CP-023

CP-024			
Propósito	Añadir un *learning-object* al *environment* de una *learning-activity*.		
Requisitos vinculados	RF-014, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-023	Archivo	pokeAddLAEnvironment.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* actualiza su *environment*. Se queda con dos *service* y un *learning-object*. Inicialmente tiene dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.24 Caso de prueba CP-024

CP-025			
Propósito	Añadir un *learning-object* al *environment* de una *learning-activity*.		
Requisitos vinculados	RF-014, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddLAEnvironment2.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* actualiza su *environment*. Se queda con un *learning-object*. Inicialmente estaba vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.25 Caso de prueba CP-025

CP-026			
Propósito	Añadir un *service* al *environment* de una *learning-activity*.		
Requisitos vinculados	RF-015, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddLAEnvironment3.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* actualiza su *environment*. Se queda con un *service*. Inicialmente estaba vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.26 Caso de prueba CP-026

CP-027			
Propósito	Añadir un *learning-object* al *environment* de una *support-activity*.		
Requisitos vinculados	RF-016, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-029	Archivo	pokeAddSAEnvironment2.zip
Resultado	OK	Fecha	27/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* actualiza su *environment*. Se queda con dos *service* y un *learning-object*. Inicialmente tenía dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.27 Caso de prueba CP-027

CP-028			
Propósito	Añadir un *service* al *environment* de una *support-activity*.		
Requisitos vinculados	RF-017, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddSAEnvironment.zip
Resultado	Fallo	Fecha	15/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* actualiza su *environment*. Se queda con dos *service*. Inicialmente tenía un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	No se ve en el reproductor.		

Tabla 8.28 Caso de prueba CP-028

CP-029			
Propósito	Añadir un *service* al *environment* de una *support-activity*.		
Requisitos vinculados	RF-017, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-028	Archivo	pokeAddSAEnvironment.zip
Resultado	OK	Fecha	27/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* actualiza su *environment*. Se queda con dos *service*. Inicialmente tenía un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.29 Caso de prueba CP-029

CP-030			
Propósito	Añadir un *learning-object* al *environment* de una *activity-structure*.		
Requisitos vinculados	RF-018, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddASEnvironment2.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *activity-structure* actualiza su *environment*. Se queda con dos *service* y un *learning-object*. Inicialmente tenía dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.30 Caso de prueba CP-030

CP-031			
Propósito	Añadir un *service* al *environment* de una *activity-structure*.		
Requisitos vinculados	RF-019, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddASEnvironment.zip
Resultado	Fallo	Fecha	15/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* actualiza su *environment*. Se queda con un *service*. Inicialmente estaba vacío. El cambio es apreciable en el reproductor.		
Fallo producido	No se ve en el reproductor.		

Tabla 8.31 Caso de prueba CP-031

CP-032			
Propósito	Añadir un *service* al *environment* de una *activity-structure*.		
Requisitos vinculados	RF-019, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-031	Archivo	pokeAddASEnvironment.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* actualiza su *environment*. Se queda con un *service*. Inicialmente estaba vacío. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.32 Caso de prueba CP-032

CP-033			
Propósito	Añadir una *learning-activity* a una *activity-structure*.		
Requisitos vinculados	RF-020, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeAddASLearning.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* actualiza sus actividades. Se queda con dos *learning-activity*. Inicialmente tenía una *learning-activity*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.33 Caso de prueba CP-033

CP-034			
Propósito	Añadir una *support-activity* a una *activity-structure*.		
Requisitos vinculados	RF-021, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-033	Archivo	pokeAddASSupport.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* actualiza sus actividades. Se queda con dos *learning-activity* y una *support-activity*. Inicialmente tenía dos *learning-activity*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.34 Caso de prueba CP-034

8.1.3 CASOS DE PRUEBA DE BORRADO

CP-035			
Propósito	Borrar un *learning-object* del *environment* de una *learning-activity*.		
Requisitos vinculados	RF-022, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-024	Archivo	pokeDelLAEnvironment.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *learning-activity* actualiza su *environment*. Se queda con dos *service*. Inicialmente tenía dos *service* y un *learning-object*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.35 Caso de prueba CP-035

CP-036			
Propósito	Borrar un *service* del *environment* de una *learning-activity*.		
Requisitos vinculados	RF-023, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-026	Archivo	pokeDelLAEnvironment2.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *learning-activity* actualiza su *environment*. Se queda vacío. Inicialmente tenía un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.36 Caso de prueba CP-036

CP-037			
Propósito	Borrar un *learning-object* del *environment* de una *support-activity*.		
Requisitos vinculados	RF-024, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-027	Archivo	pokeDelSAEnvironment2.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* actualiza su *environment*. Se queda con un *service*. Inicialmente tenía un *service* y un *learning-object*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.37 Caso de prueba CP-037

CP-038			
Propósito	Borrar un *service* del *environment* de una *support-activity*.		
Requisitos vinculados	RF-025, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeDelSAEnvironment.zip
Resultado	OK	Fecha	28/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *support-activity* actualiza su *environment*. Se queda vacío. Inicialmente tenía un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.38 Caso de prueba CP-038

CP-039			
Propósito	Borrar un *learning-object* del *environment* de una *activity-structure*.		
Requisitos vinculados	RF-026, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-030	Archivo	pokeDelASEnvironment3.zip
Resultado	OK	Fecha	29/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *activity-structure* actualiza su *environment*. Se queda con dos *service*. Inicialmente tenía dos *service* y un *learning-object*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.39 Caso de prueba CP-039

CP-040			
Propósito	Borrar un *service* del *environment* de una *activity-structure*.		
Requisitos vinculados	RF-027, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeDelASEnvironment.zip
Resultado	OK	Fecha	29/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *activity-structure* actualiza su *environment*. Se queda con un *service*. Inicialmente tenía dos *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.40 Caso de prueba CP-040

CP-041			
Propósito	Borrar un *service* del *environment* de una *activity-structure*.		
Requisitos vinculados	RF-027, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	CP-040	Archivo	pokeDelASEnvironment2.zip
Resultado	OK	Fecha	29/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *activity-structure* actualiza su *environment*. Se queda vacío. Inicialmente tenía un *service*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.41 Caso de prueba CP-041

CP-042			
Propósito	Borrar una *learning-activity* de una *activity-structure*.		
Requisitos vinculados	RF-028, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeDelASLearning.zip
Resultado	OK	Fecha	26/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario alumno.		
Salida	La *activity-structure* actualiza sus actividades. Se queda con tres *learning-activity*. Inicialmente tenía cuatro *learning-activity*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.42 Caso de prueba CP-042

CP-043			
Propósito	Borrar una *support-activity* de una *activity-structure*.		
Requisitos vinculados	RF-029, RF-030 RNFO-001, RNFO-002, RNFO-003, RNFO-004, RNFO-005		
Dependencias	-	Archivo	pokeDelASSupport.zip
Resultado	OK	Fecha	26/12/09
Entrada	Lanzar CCRT. Publicar una UOL. Navegar a través de Clicc indicando la UOL, la ejecución y el usuario tutor.		
Salida	La *activity-structure* actualiza sus actividades. Se queda vacío. Inicialmente tenía una *support-activity*. El cambio es apreciable en el reproductor.		
Fallo producido	-		

Tabla 8.43 Caso de prueba CP-043

8.2 MATRIZ DE TRAZABILIDAD

Una vez que se han ejecutado todos los casos de prueba y se ha comprobado que su resultado es el esperado, se puede afirmar que el sistema cumple con los objetivos planteados y con los requisitos especificados por el usuario. A continuación se muestra la matriz de trazabilidad entre los casos de prueba y los requisitos, donde se puede comprobar qué caso de prueba satisface cada requisito.

	CP-001	CP-002	CP-003	CP-004	CP-005	CP-006	CP-007
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-001	X	X	X	X	X	X	X
RF-030	X	X	X	X	X	X	X

Tabla 8.44 Matriz de trazabilidad para los casos de prueba CP-001 a CP-007

	CP-008	CP-009	CP-010	CP-011	CP-012	CP-013	CP-014
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-002	X	X					
RF-003			X	X	X		
RF-004						X	
RF-005							X
RF-030	X	X	X	X	X	X	X

Tabla 8.45 Matriz de trazabilidad para los casos de prueba CP-008 a CP-014

	CP-015	CP-016	CP-017	CP-018	CP-019	CP-020	CP-021
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-006	X						
RF-007		X					
RF-008			X				
RF-009				X			
RF-010					X		
RF-011						X	
RF-012							X
RF-030	X	X	X	X	X	X	X

Tabla 8.46 Matriz de trazabilidad para los casos de prueba CP-015 a CP-021

	CP-022	CP-023	CP-024	CP-025	CP-026	CP-027	CP-028
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-013	X						
RF-014		X	X	X			
RF-015					X		
RF-016						X	
RF-017							X
RF-030	X	X	X	X	X	X	X

Tabla 8.47 Matriz de trazabilidad para los casos de prueba CP-022 a CP-028

	CP-029	CP-030	CP-031	CP-032	CP-033	CP-034	CP-035
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-017	X						
RF-018		X					
RF-019			X	X			
RF-020					X		
RF-021						X	
RF-022							X
RF-030	X	X	X	X	X	X	X

Tabla 8.48 Matriz de trazabilidad para los casos de prueba CP-029 a CP-035

	CP-036	CP-037	CP-038	CP-039	CP-040	CP-041	CP-042
RNFO-001	X	X	X	X	X	X	X
RNFO-002	X	X	X	X	X	X	X
RNFO-003	X	X	X	X	X	X	X
RNFO-004	X	X	X	X	X	X	X
RNFO-005	X	X	X	X	X	X	X
RF-023	X						
RF-024		X					
RF-025			X				
RF-026				X			
RF-027					X	X	
RF-028							X
RF-030	X	X	X	X	X	X	X

Tabla 8.49 Matriz de trazabilidad para los casos de prueba CP-036 a CP-042

	CP-043
RNFO-001	X
RNFO-002	X
RNFO-003	X
RNFO-004	X
RNFO-005	X
RF-029	X
RF-030	X

Tabla 8.50 Matriz de trazabilidad para el caso de prueba CP-043

9 CONCLUSIONES

En este apartado se recogen las diferentes conclusiones extraídas a lo largo de la realización de este Proyecto Fin de Carrera. Se hablará sobre las aportaciones que ofrece este trabajo, sobre los principales problemas encontrados, sobre futuras líneas de trabajo y sobre las aportaciones personales que me ha ofrecido este proyecto.

Este Proyecto Fin de Carrera surge debido a las carencias existentes en materia de *software* que implemente procesos de aprendizaje especificados mediante IMS *Learning Design* y, más concretamente, en lo referido a las adaptaciones en tiempo de ejecución para dichos procesos.

Actualmente, para introducir cambios en una UOL publicada y en ejecución, es necesario parar el proceso y rediseñarla. El principal objetivo que ha cumplido este proyecto es el de desarrollar un módulo de adaptación que permita introducir modificaciones sin la necesidad de detener el flujo de trabajo en ejecución. Supóngase el caso en que una serie de alumnos están desarrollando su trabajo y el tutor que los guía decide introducir nuevo material de apoyo para alguna de las actividades en que están trabajando. Este cambio supondría la detención del proceso y la repetición del trabajo ya realizado. Este problema ha quedado resuelto.

Hay que destacar varias dificultades encontradas: una es el uso de estándares y la otra es que se ha desarrollado una extensión sobre una herramienta ya existente y carente de documentación.

El uso de estándares (IMS LD) se considera como una dificultad debido a que ha implicado un estudio previo para su comprensión y el futuro desarrollo del módulo de adaptación. Pero también cabe mencionar que supone un factor de calidad, ya que se consigue interoperabilidad.

La principal dificultad encontrada ha sido la familiarización con el motor de ejecución CopperCore. Esto se debe a que su código fuente está compuesto por más de 370 clases, lo que unido a la falta de documentación del mismo, ha supuesto un gran esfuerzo en el estudio previo a la implementación del módulo de adaptación desarrollado. Si bien no se han usado todas las clases disponibles, y por tanto no ha sido necesario estudiarlas todas, el código está compuesto por diferentes paquetes, cada uno con una funcionalidad definida y con varias interfaces. Por tanto, seguir el camino del código y entender qué hace cada clase ha sido una tarea ardua y pesada que ha producido un desvío importante entre la planificación original y la planificación real.

Este módulo de adaptación puede ser aprovechado para mejorar otras características ofrecidas por IMS LD y que no han sido tenidas en cuenta en este Proyecto Fin de Carrera, como por ejemplo la monitorización de las actividades llevadas a cabo por los diferentes usuarios implicados en un proceso de aprendizaje. Por otro lado, la forma de interaccionar con CopperCore actualmente es a través de una consola de comandos. Una

futura línea de trabajo podría integrar este módulo de adaptación con el editor de cursos de IMS LD ya existente y mencionado en este Proyecto Fin de Carrera, de manera que en vez de tener que subir un *Adaptation Poke* (generado por esta herramienta) a CopperCore manualmente, al realizar los cambios en el editor estos se aplicaran directamente, eliminando de esta forma la tediosa tarea de lidiar con una consola de comandos.

Hay que mencionar que el esfuerzo llevado a cabo y el tiempo dedicado para la realización de este Proyecto Fin de Carrera han sido muy grandes, ya que ha supuesto un pequeño cambio con respecto al conjunto de asignaturas que han compuesto mi Carrera, debido principalmente a que he tenido que luchar contra la falta de plazos fijos y a que el trabajo se ha realizado de forma individual.

La principal aportación personal que me ha producido este Proyecto Fin de Carrera proviene del trabajo individual desarrollado, ya que durante los últimos años, en las diferentes asignaturas que he cursado, las prácticas se han desarrollado bien en parejas, bien en grupos de tres o cuatro personas. Este trabajo personal me ha aportado una visión referente al esfuerzo de trabajar individualmente.

Por otra parte, aunque el entorno en que se desarrolla este Proyecto Fin de Carrera (estándar IMS LD, procesos de aprendizaje, etc.) ha sido nuevo para mí, me ha resultado agradable e interesante obtener una visión del mundo del *eLearning* que antes no poseía.

Otra aportación también destacable consiste en el hecho de haberme encontrado con una herramienta ya desarrollada y sin documentación, que ha supuesto enfrentarme a un código implementado por otros programadores. Según me han comentado, este problema es muy frecuente en el mundo laboral al que aspiro a entrar y, por tanto, espero me haya servido como un complemento a la formación adquirida durante estos años en la Universidad Carlos III de Madrid.

ACRÓNIMOS

- ADL: *Advanced Distributed Learning.*
- AICC: *Aviation Industry Computer-Based Training Comitee.*
- AIES: *Adaptive and Intelligent Educational Systems.*
- API: *Application Programming Interface.*
- ASI: *Assessments Sections Items.*
- BMP: *Bean Managed Persistence.*
- CBT: *Computer-Based Training.*
- CCRT: *CopperCore Run Time.*
- CCSI: *CopperCore Service Integration.*
- CD-ROM: *Compact Disc – Read Only Memory.*
- CLICC: *Command Line Interface CopperCore*.
- CMS: *Content Management Systems.*
- CORBA: *Common Object Request Broker Architecture*.
- EJB: *Entreprise JavaBeans.*
- EML: *Educational Modelling Languages*.
- GNU GPL: *GNU General Public License.*
- HSQLDB: *Hyperthreaded Structured Query Language Database*.
- IDE: *Integrated Development Environment.*
- IMS LD: *Instructional Management System Learning Design.*
- IMS QTI: *Instructional Management System Question & Test Interoperability.*
- ISO: *International Organization for Standarization*.

- ITS: *Intelligent Tutoring Systems.*
- IVA: Impuesto sobre el Valor Añadido.
- J2EE: *Java 2 Enterprise Edition.*
- J2SE: *Java 2 Standard Edition.*
- JDBC: *Java DataBase Connectivity.*
- JDK: *Java Development Kit.*
- JMS: *Java Message Service.*
- LAMS: *Learning Activity Management System.*
- LCMS: *Learning Content Management Systems.*
- LDL: *Learning Design Language.*
- LMS: *Learning Management Systems.*
- LOM: *Learning Object Metadata.*
- MOODLE: *Modular Object-Oriented Dynamic Learning Environment.*
- MS SQL: *Microsoft Structured Query Language.*
- OSX: *Operative System X.*
- OUNL: *Open Universiteit Nederland.*
- PAPI: *Public And Private Information.*
- RMI: *Remote Method Invocation.*
- SCO: *Shareable Content Object.*
- SCORM: *Shareable Content Object Reference Model.*
- SGML: *Standard Generalized Markup Language.*
- SOAP: *Simple Object Access Protocol.*

- TIC: Tecnologías de la Información y de la Comunicación.
- UNED: Universidad Nacional de Educación a Distancia.
- UML: *Unified Modeling Language.*
- UOL: *Unit Of Learning.*
- XML: *Extensible Markup Language.*
- XSL: *Extensible Stylesheet Language.*
- W3C: *World Wide Web Consortium*.

BIBLIOGRAFÍA

[1] LTSC, "Learning Technology Standards Committee", disponible en http://ltsc.ieee.org/.

[2] AICC, "Aviation Industry CBT Comittee", disponible en http://www.aicc.org/.

[3] IMS Global Learning Consortium, "Instructional Management Systems", disponible en http://imsproject.org/.

[4] ADL, "Advanced Distributed Learning", disponible en http://www.adlnet.org/.

[5] IEEE Learning Technologies Standards Committee (2002), "Learning Objects Meta-Data Specification. Version 6.3", disponible en http://ltsc.ieee.org/wg12/files/.

[6] IMS Global Learning Consortium (2004), "IMS Content Packaging Information Model, Version 1.1.4 Final Specification", disponible en http://www.imsglobal.org/content/packaging/cpv1p1p4/imscp_infov1p1p4.html.

[7] IMS Global Learning Consortium (2006), "IMS Question & Test Interoperability Overview Version 2.1 Public Draft (revision 2) Specification", disponible en http://www.imsglobal.org/question/index.html.

[8] IMS Global Learning Consortium (2001), "IMS Learner Information Package Information Model. Version 1.0 final specification", disponible en http://www.imsglobal.org/profiles/index.html.

[9] ARIADNE, "Alliance of Remote Instructional Authoring and Distribution Networks for Europe", disponible en http://ariadne2.unil.ch/.

[10] IMS Global Learning Consortium (2003), "IMS Simple Sequencing Information and Behavior Model. Version 1.0 Public Draft Specification", disponible en http://www.imsglobal.org/simplesequencing/ssv1p0/imsss_infov1p0.html.

[11] IMS Global Learning Consortium, "IMS Global Cartridge", disponible en http://www.imsglobal.org/commoncartridge.html.

[12] LAMS International, "Learning Activity Management System", disponible en http://www.lamsinternational.com/.

[13] RELOAD Project, "Reusable eLearning Object Authoring & Delibery", disponible en http://www.reload.ac.uk/.

[14] COGIGRAPH Technologies - Télé-université's LICEF research center, "MOT™ / MOT Plus™ - Knowledge Modeling Editor", disponible en http://www.cogigraph.com/Produits/MOTetMOTplus/tabid/995/language/en-US/Default.aspx.

[15] Martens, H., Vogten, H., Rosmalen, P. v. y Koper, E. J. R. (2004), "Coppercore", from SourceForge, disponible en http://coppercore.sourceforge.net/.

[16] Service Based learning Design Player (2005), The Open University, disponible en http://sled.open.ac.uk/index.php.

[17] MOODLE Org, "MOODLE", http://moodle.org/.

[18] Zarraonandia Ayo, Telmo Agustín (2007), "Adaptaciones de unidades de aprendizaje en tiempo de ejecución", tesis doctoral, disponible en http://e-archivo.uc3m.es:8080/dspace/handle/10016/2465.

[19] J2EE, Sun Microsystems, disponible en http://java.sun.com/j2ee/setstandard.html.

[20] Apache Ant, The Apache Ant Project, disponible en http://ant.apache.org/.

[21] Eclipse, Eclipse Foundation, disponible en http://www.eclipse.org/.

[22] CopperCore, CopperCore API Javadoc, disponible en http://coppercore.sourceforge.net/documentation/javadoc/coppercore/index.html.

[23] CopperCore, CopperCore SOAP API Javadoc, disponible en http://coppercore.sourceforge.net/documentation/javadoc/soap/index.html.

[24] CopperCore, XML's schemas, disponible en http://coppercore.sourceforge.net/documentation/schemas/index.shtml.

[25] CopperCore, CopperCore Run Time environment, disponible en http://coppercore.sourceforge.net/downloads.shtml.

[26] J2EE SDK 1.4, Sun Microsystems, disponible en http://java.sun.com/j2ee/1.4/download.html.

GLOSARIO DE TÉRMINOS

- E-Learning: sistema de educación electrónico o a distancia en el que se integra el uso de las tecnologías de la información y otros elementos pedagógicos para la formación, capacitación y enseñanza de los usuarios o estudiantes en línea.

- Feedback: también denominado retroalimentación, consiste en un proceso en el que la señal de salida es redirigida a la señal de entrada, consiguiendo de esta manera producir cambios dependiendo de distintos factores.

- Framework: estructura conceptual y tecnológica de soporte definida normalmente con artefactos de *software* concretos, mediante la cual otro proyecto de *software* puede ser organizado y desarrollado.

- Hardware: se refiere al conjunto de los componentes que integran la parte material de un ordenador.

- Online: conectado a una red o sistema mayor.

- Open source: también conocido como código abierto, hace referencia al *software* distribuido y desarrollado libremente.

- Plugin: también conocido como complemento, es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

- Router: también conocido como enrutador, es un dispositivo para la interconexión de redes informáticas.

- Script: es un guión o conjunto de instrucciones que permite la automatización de tareas.

- Software: referido al equipamiento lógico o soporte lógico de una computadora digital, y comprende el conjunto de componentes lógicos necesarios para hacer posible la realización de una tarea específica.

- Web: también conocido como World Wide Web, es el sistema de documentos interconectados por enlaces de hipertexto disponibles en Internet.

- Website: también conocido como sitio Web, es un conjunto de páginas Web.